(2)

Image Compression Using Fractals and Wavelets

Final Report for the Phase II Contract

Sponsored by the Office of Naval Research

Contract No. N00014-91-C-0117

June 2, 1993

Submitted To
Scientific Officer
Dr. Michael F. Shlesinger
Attn: MFS, Code: 1112
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

NETROLOGIC Inc. • 5080 Shoreham Place, Suite 201 • San Diego, CA 92122
Phone: (619) 625-6255 • Fax: (619) 625-6258

Image Compression Using Fractals and Wavelets

Final Report for the Phase II Contract

Sponsored by the Office of Naval Research

Contract No. N00014-91-C-0117

June 2, 1993

Statement A per telecon
Michael Shlesinger ONR/Code 1112
Arlington, VA 22217-5000

NWW 6/17/93

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

Submitted To

Scientific Officer
Dr. Michael F. Shlesinger
Attn: MFS, Code: 1112
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

NETROLOGIC Inc. • 5080 Shoreham Place, Suite 201 • San Diego, CA 92122
Phone: (619) 625-6255 • Fax: (6'9) 625-6258

Acknowledgement

Mr. Dan Greenwood served as the project Principal Investigator. Dr. Albert Lawrence guided the theoretical development of the Wavelet Compression study and Dr. Yuval Fisher guided the Fractal Compression study. Mr. Fareed Stevenson, Mr. Richard Rea and Mr. Yaokun Zhong provided software support. The encouragement and supervision of ONR's Dr. Shlesinger is greatly appreciated by all the participants of this study.

Abstract

There are several approaches to image compression. The current most popular methods relies on eliminating high frequency components of the signal by storing only the low frequency Fourier coefficients. Other methods use a "building block" approach, breaking up images into a small number of canonical pieces and storing only a reference to which piece goes where. Our research has focused on a new scheme based on fractals. Our approach to image compression is to tessellate the image with a tiling which varies with the local image complexity, and to check for self similarity amongst the tiles. Self similarities are coded as systems of affine transformations which can be stored far more compactly than the original images on small platforms.

An original objective of our Phase II research project was to develop a hardware implementation of our Fractal based algorithm and to investigate various techniques for encoding. During the course of initial investigations it became apparent that the bulk of our efforts should be directed toward speeding up the software algorithms, especially in the decoding and adapting our fractal encoding methods to color images. This is because the most significant commercial applications of image compression require fast decoding.

Because the fractal compression method has not been entirely satisfactory for some types of images such as maps, fingerprints and satellite images we have implemented a version of the wavelet compression method reported by Antonini.

This encoding method has three main steps: a wavelet transform followed by a lattice vector quantization followed by a Huffman encoding of the output vectors. The Huffman code step is not reported in the report by Antonini. A standard codebook may be used for transmission, with a provision for sending codebook entries for rarely-encountered vectors.

MATERIAL INSPECTION AND RECEIVING REPORT

Form Approved
OMB No. 0704-0248

1. PROC. INSTRUMENT IDEN. (CONTRACT) N00014-91-C-0117	(ORDER) NO.	6. INVOICE NO./DATE 3 JUN 1993	7. PAGE 1 OF 1	8. ACCEPTANCE POINT D
2. SHIPMENT NO. NET0001Z	3. DATE SHIPPED 3 JUN 93	4. B/L TCN	5. DISCOUNT TERMS N/A	

9. PRIME CONTRACTOR CODE 0B9N7 NETROLOGIC Inc. 5080 Shoreham Place, Suite 201 San Diego, CA 92122-5932	10. ADMINISTERED BY CODE S0514A DCASMA San Diego 7675 Dagget Street, Suite 200 San Diego, CA 92111-2241
11. SHIPPED FROM (If other than 9) CODE FOB:	12. PAYMENT WILL BE MADE BY CODE SC1008 DFC/San Francisco P.O. Box 182380 Columbus, OH 43218-2380
13. SHIPPED TO CODE N00014 Scientific Officer, M. Shlesinger Attn: MFS, Code:1112 Office of Naval Research, 800 N. Quincy St Arlington, VA 22217-5000	14. MARKED FOR CODE N00014 M. Shlesinger Attn: MFS, Code:1112

15. ITEM NO.	16. STOCK/PART NO. DESCRIPTION (Indicate number of shipping containers - type of container - container number.)	17. QUANTITY SHIP/REC'D *	18. UNIT	19. UNIT PRICE	20. AMOUNT
0001	Personnel and Facilities to conduct the work titled "Fractal Image Encoding"	LOT	EA	454,968	454,968
0002	Reports and Data in Accordance with Exhibit A of Contract	LOT	EA	NSP	NSP
A002	Final Report titled "Image Compression Using Fractals and Wavelets"				

21. CONTRACT QUALITY ASSURANCE

A. ORIGIN	B. DESTINATION	22. RECEIVER'S USE
☐ CQA ☐ ACCEPTANCE of listed items has been made by me or under my supervision and they conform to contract, except as noted herein or on supporting documents.	☐ CQA ☐ ACCEPTANCE of listed items has been made by me or under my supervision and they conform to contract, except as noted herein or on supporting documents.	Quantities shown in column 17 were received in apparent good condition except as noted.
DATE / SIGNATURE OF AUTH GOVT REP	DATE / SIGNATURE OF AUTH GOVT REP	DATE RECEIVED / SIGNATURE OF AUTH GOVT REP
TYPED NAME AND OFFICE	TYPED NAME AND TITLE	TYPED NAME AND OFFICE
		If quantity received by the Government is the same as quantity shipped, indicate by (✓) mark, if different, enter actual quantity received below quantity shipped and encircle.

23. CONTRACTOR USE ONLY

If there are questions or problems concerning this document, please contact
Sarah Bode
Netrologic Inc.
(619) 625-6255

Table of Contents

Figures and Tables

1.0 Aims of the Project

With the advance of the information age the need for mass information storage and retrieval grows. The capacity of commercial storage devices, however, has not kept pace with the proliferation of image data. The coding, storage, and reconstruction of images is a major concern in many applications of computer technology to technical and scientific problems. One example is the flood of geophysical and intelligence data originating from satellite platforms. In such applications it is highly desirable to reduce the storage and transmission requirements for image data. The applications for such a technology are innumerable. The increasing need for image storage challenges the ability of technological innovation to provide solutions. As a typical example, consider mass storage devices of personal computers. Such devices can store roughly 200 MBytes of data; but this provides storage space for only 200 images of size 1024 x 1024 pixels at 8 bits per pixel (bbp). Utilizing a compression scheme which provides 40:1 compression allows 8,000 images to be stored - a significant improvement.

Image compression translates directly to cost savings in the commercial world also. Although the storage cost per bit of current commercial devices is currently about half a millionth of a dollar, a photo album with several hundred photos can cost over a thousand dollars to store! This is one area is which image compression can play an important role. Storing the images in less memory leads to a direct reduction in cost. Another useful feature of image compression is the rapid transmission of data; less data requires less time to send.

How can image data be compressed? Most data contains some amount of redundancy, which can sometimes be removed for storage and replaced for recovery, but this redundancy does not lead to high compression. Fortunately, the human eye is not sensitive to a variety of types of information loss. The image can be changed in many ways that are either not detectable by the human eye or do not contribute to "degradation" of the image. If these changes are made so that the data becomes highly redundant, then the data can be compressed.

There are several approaches to image compression. The current most popular methods relies on eliminating high frequency components of the signal by storing only the low frequency Fourier coefficients. Other methods use a "building block" approach, breaking up images into a small number of canonical pieces and storing only a reference to which piece goes where. Our research has focused on a new scheme based on fractals. Our approach to image compression is to tessellate the image with a tiling which varies with the local image complexity, and to check for self similarity amongst the tiles. Self similarities are coded as systems of affine transformations which c n be stored far more compactly than the original images on small platforms.

A secondary objective of our research was to investigate the relationship between affine transforms and wavelets. This investigation let to the development of an alternate algorithm.

2.0 Technical Objectives

Fractals

An original objective of our Phase II research project was to develop a hardware implementation of our Fractal based algorithm and to investigate various techniques for encoding. During the course of initial investigations it became apparent that the bulk of our efforts should be directed toward speeding up the software algorithms, especially in the decoding and adapting our fractal encoding methods to color images. This is because the most significant commercial applications of image compression require fast decoding.

3.0 Background

The development of the line of research which has culminated in the so-called fractal data compression methods begins with Hutchinson. Hutchinson [16] introduced the theory of iterated functions system (a term coined by Barnsley) to model self similar sets (such as in Figure 3-3). Demko, Hodges and Naylor [10] first suggested using iterated function systems to model complex objects in computer graphics. Barnsley, Demko, Elton, Sloan and others generalized the concepts and suggested the use of fractals to model "natural scenes". In his thesis [17] A. Jacquin developed an image encoding scheme based on iterated Markov operators on measure space and used it to encode 6 bit/pixel monochrome images.

An improved version of this scheme along with other schemes can be found in work done by Fisher in [11], [12], and [18]. Much of the following fractal theoretical (also Section 4) material is also covered in many of the recent reports of Y. Fisher resulting from his efforts at UCSD, NOSC and NETROLOGIC.

We begin by describing a simple scheme that can generate complex looking fractals from a small amount of information. Next we will generalize this scheme to allow the encoding of images as "fractals", and finally we will discuss some of the ways this scheme can be implemented.

Imagine a special type of copying machine that reduces the image to be copied by a half and reproduces it three time on the copy. Figure 3-1 shows this. Figure 3-2 shows several iterations of this process on several input images. What we observe is that all the copies seem to be converging to the same final image, the one in Figure 3-2(c). We call this image the *attractor* for this process. Because the copying machine reduces the input image, any initial image will be reduced to a point as we repeatedly run the machine. Thus, the initial image placed on the copying machine doesn't effect the final attractor; in fact, it is only the position and the orientation of the copies that determines what the final image will look like.

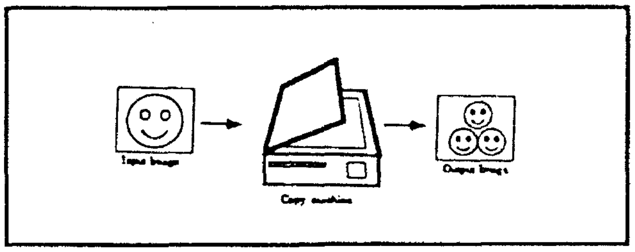


Figure 3-1 A copy machine that makes three reduced copies of the input image.

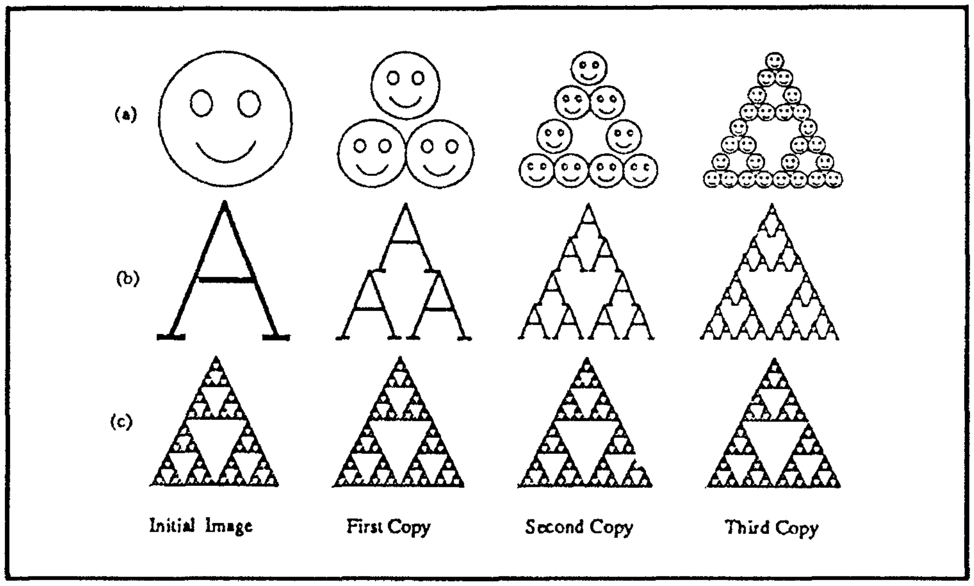


Figure 3-2 The first three copies generated on the copying machine of Figure 3-1.

Because the transforms on the input image determine the final result of running the copy machine in a feedback loop, we need only describe these transformations. Different transformations lead to different attractors, with the technical limitation that the transformations must be contractive - that is, a given transformation applied to any two points in the input image must bring them closer together in the copy.

Contractive Transforms

A transformation w is said to be contractive if for any two points P_1, P_2, the distance

$$d\,(w\,(P_1), w\,(P_2)) < s d\,(P_1, P_2\,) \tag{3-1}$$

for some $s < 1$. This formula says the application of a contractive map always brings points closer together (by some factor less than 1). This definition is completely general, applying to any space on this we can define a distance function $d(P_1,P_2)$. In our case, we work in the plane, so that if the points have coordinates $P_1 = (x_1,y_1)$ and $P_2 = (x_2,y_2)$, then

$$d\,(P_1,\ P_1) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3-2}$$

An example of a contractive transformation of the p'ane is

$$w\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}. \tag{3-3}$$

which halves the distance between any two points.

Contractive transformations have the nice property that when they are repeatedly applied, they converge to a point which remains fixed upon further iteration. For example, the map w above applied to any initial point (x,y) will yield the sequence of point $(x/2, y/2), (x/4, y/4),...$ which can be seen to converge to the point $(0,0)$ which remains fixed.

This technical condition is very natural, since if points in the copy were spread out the attractor would have to be of infinite size. Except for this condition, the transformations can have any form. In practice, choosing transformations of the form

$$w_i\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix} \tag{3-4}$$

is sufficient to yield a rich and interesting set of attractors. Such transformations are called affine transformations of the plane, and each can skew, stretch, rotate, scale and translate an input image; in particular, affine transformations always map squares to parallelograms.

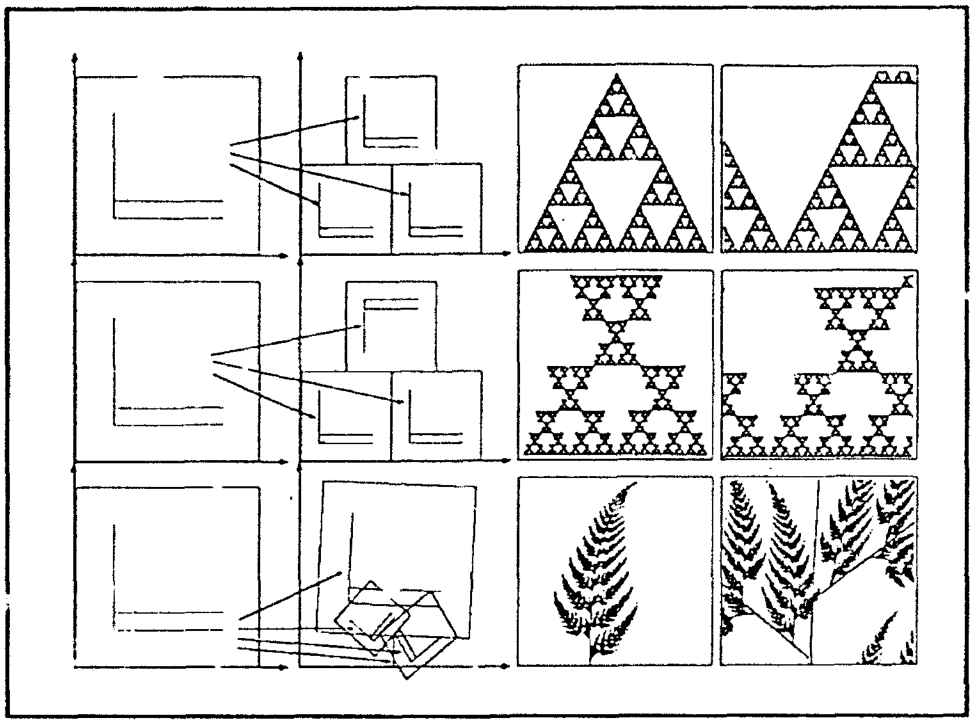


Figure 3-3 Transformations, their attractor, and a zoom on the attractor.

A common feature of all attractors formed this way is that in the position of each of the images of the original square on the left there is a transformed copy of the whole image. Thus, each image is formed from transformed (and reduced) copies of itself, and hence it must have detail at every scale. That is, the images are fractals. This method of generating fractals is due to John Hutchinson [16], and more information about many ways to generate such fractals can be found in books by Barnsley [3] and Peitgen, Saupe, and Jurgens [21,22].

Barnsley suggested that perhaps storing images as collections of transformations could lead to image compression. His argument went as follows: the fern in Figure 3-3 looks complicated and intricate, yet it is generated from only 4 affine transformations. Each affine transformation w_i is defined by 6 numbers, a_i, b_i, c_i, d_i, e_i and f_i which do not require much memory to store on a computer (they can be stored in 4 transformations x 6 number/transformation x 32 bits/number = 768 bits). Storing the image of the fern as a collections of pixels, however, requires much more memory (at least 65,636 bits for the resolution shown in Figure 3-3). So if we wish to store a picture of a fern, then we can do it by storing the numbers that define the affine transformations and simply generate the fern when we want to see it. This works with images which are a special type of fractal to begin with. Suppose that we were given any arbitrary image, say a face. If a small number of affine transformation could generate that face, then it too could be stored compactly. The trick is finding those numbers. Unfortunately we can only approximate an image in this way.

Before we discuss the problem of approximating an image by a system of affine transforms we will set up some machinery for describing these systems.

Iterated Function Systems

Running the special copy machine in a feedback loop is a metaphor for a mathematical model called an iterated function system (IFS). An iterated function system consists of a collection of contractive transformations $\{w^{i}: \frac{\geq^{2}}{<} \rightarrow R^{2} \mid i = 1,\ldots n\}$ which map the plane R^2 to itself. This collection of transformations defines a map

$$W(\cdot) = \bigcup_{i=1}^{n} w_i(\cdot) \tag{3-5}$$

The map W is not applied to the plane, it is applied to sets - that is, collections of points in the plane. Given an input set S, we can compute $w_i(S)$ for each i, take the union of these sets, and get a new set $W(S)$. So W is a map on the space of subsets of the plane. We will call a subset of the plane an image, because the set defines an image when the points in the set are drawn in black, and because later we will want to use the same notation on graphs of functions which will represent actual images. An important fact proved by Hutchinson is that when the w_i are contractive in the plane, then W is contractive in a space of (closed and bounded) subsets of the plane. Hutchinson's theorem allows us to use the contractive mapping fixed point theorem which tells us that the map W will have a unique fixed point in the space of all images. That is, whatever image (or set) we start with, we can repeatedly apply W to it and we will converge to a fixed image. Thus W (or the w_i) completely determine a unique image.

In other words, given an input image f_0, we can run the copying machine once to get $f_1 = W(f_0)$, twice to get $f_2 = W(f_1) = W(W(f_0)) \equiv W^{\circ 2}(f_0)$, and so on. The attractor, which is the result of running the copying machine in a feedback loop, is the limit set

$$|W| \equiv f_{\infty} = \lim_{n \to \infty} W^{\circ n}(f_0) \tag{3-6}$$

which is not dependent on the choice of f_0. Iterated function systems are interesting in their own right, but we are not concerned with them specifically. We will generalize the idea of the copy machine and use it to encode grey-scale images; that is, images that are not just black and white but which contain shades of grey as well.

The Contractive Mapping Fixed Point Theorem

The contractive mapping fixed point theorem says that something that is intuitively obvious: if a map is contractive then when we apply it repeatedly starting with any initial point we converge to a unique fixed point. for example, the map $w(x) = x/2$ on the real line is contractive for the normal metric $d(x,y) = |x-y|$, because the distance between $w(x)$ and $w(y)$ is half the distance between x and y. Furthermore, if we iterate w from any initial point x, we get a sequence of points $x/2$, $x/4$,... that converges to the fixed point 0.

This theorem tells us when we can expect a collection of transformations to define images. Let's write it precisely and examine it carefully.

THE CONTRACTIVE MAPPING FIXED POINT THEOREM. *If X is a complete metric space and $W : X \rightarrow X$ is contractive, then W has a unique fixed point $|W|$.*

A complete metric space is a "gap-less" space on which we can measure the distance between any two points. For example, the real line is a complete metric space with distance between any two points x and y given by $|x-y|$. The set of all fractions of integers, however, is not complete. We can measure the distance between two fractions in the same way, but between any two elements of the space we find a real number (that is, a "gap") which is not a fraction and hence is not in the space. Returning to our example, the map w can operate on the space of fractions, however the map $x \mapsto \frac{1}{\pi} x$ cannot. This map is contractive, but after one application of the map we are no longer in the same space we began in. This is the reason for requiring that we work in a complete metric space.

A fixed point $|W| \in X$ of W is a point that satisfies $W(|W|) = |W|$. Our mapping $w(x) = x/2$ on the real line has a unique fixed point: Start with an arbitrary point $x \in X$. Now iterate W to get a sequence of points $x, W(x), W(W(x),...$ The distance between $W(x)$ and $W(W(x))$ is less by some factor $s < 1$ than the distance between x and $W(x)$. At each step the distance to the next point is less by some factor than the distance to the previous point. Because we are taking geometrically smaller steps, and since our space has no gaps, we must eventually converge to a point in the space which we denote $|W| = lim_{n \rightarrow \infty} W^{\circ n}(x)$. This point is fixed, because applying W one more time is the same as starting at $W(x)$ instead of x, and either way we get to the same point.

The fixed point is unique because if we assume that there are two, then we will get a contradiction: Suppose there are two fixed points x_1 and x_2; then the distance between $W(x_1)$ and $W(x_2)$, which is the distance between x_1 and x_2; this is a contradiction.

Thus, the main result we have demonstrated is that when W is contractive, we get a fixed point

$$|W| = \lim_{n \to \infty} W^{\circ n}(x) \tag{3-7}$$

for any initial x.

Grey-Scale Images

The second generalization we need is a method for handling grey-scale images. To do this we need a mathematical model of an image. Figure 3-4 shows the graph of a special function $z = f(x,y)$. This graph is generated by using the image of Lena (see figure 3-5) and plotting the grey level of the pixel at position (x,y) as a height, with white being high and black being low. This is our model for an image, except that while the graph in figure 3-4 is generated by connecting the heights on a 64 x 64 grid, we generalize this and assume that every position (x,y) can have an independent height. That is, our model of an image has infinite resolution.

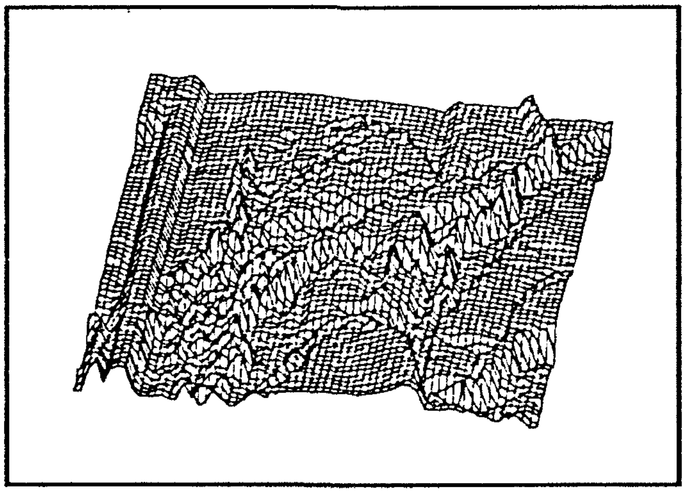


Figure 3-4 A graph generated from the Lena image.

Thus, when we wish to refer to an image, we refer to the function $f(x,y)$ which gives the grey level at each point (x,y). In practice, we will not distinguish between the *function* f (which gives us a z value for each x,y coordinate) and the graph of the function (which is a set in 3 space consisting of the points in the surface defined by f) For simplicity, we assume we are dealing with square images of size 1; that is, $(x,y) \in \{(u,v) : 0 \leq u,v \leq 1\} \equiv I^2$, and $f(x,y) \in I = [0,1]$. We have introduced some convenient notation here: I means the interval $[0,1]$ and I^2 is the unit square.

A Metric on Images

Now take the collection of all possible images: clouds, trees, dogs, the surface of Jupiter, etc. We want to find a map W which takes an input image and yields an output image, just as we did before with subsets of the plane. If we want to know when W is contractive, we will have to define a distance between two images. There are many metrics to choose from, but the simplest to use is the sup metric

$$\delta(f,g) = \sup_{(x,y) \in I^2} |f(x,y) - g(x,y)|. \tag{3-9}$$

This metric finds the position (x,y) where two images f and g differ the most and sets this value at the distance between f and g.

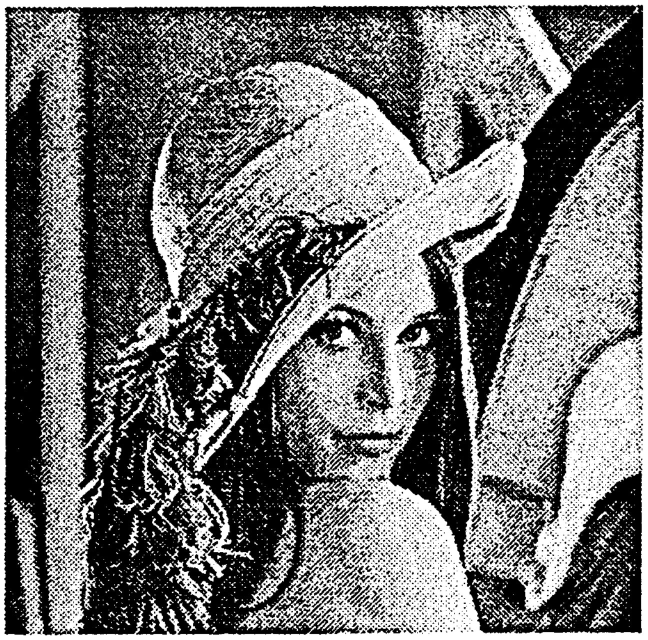


Figure 3-5 The original 256 x 256 pixel Lena image.

Natural Images are not Exactly Self Similar

A typical image of a face, for example Figure 3-5, does not contain the type of self-similarity that can be found in the fractals of Figure 3-3. The image does not appear to contain affine transformations of itself. But, in fact, this image does contain a different sort of self-similarity. Figure 3-6 shows sample regions of Lena which are similar at different scales: a portion of her shoulder overlaps a region that is almost identical, and a portion of the reflection of the hat in the mirror is similar (after transformation) to a part of her hat. The distinction from the kind of self-similarity we saw in Figure 3-3 is that rather than having the image be formed of copies of its whole self (under appropriate affine transformation), here the image will be formed of copies of properly transformed parts of itself. These transformed parts do not fit together, in general, to form an exact copy of the original image, and so we must allow some error in our representation of an image as a set of transformations. This means that the image we encode as a set of transformations will not be an identical copy of the original image but rather an approximation of it.

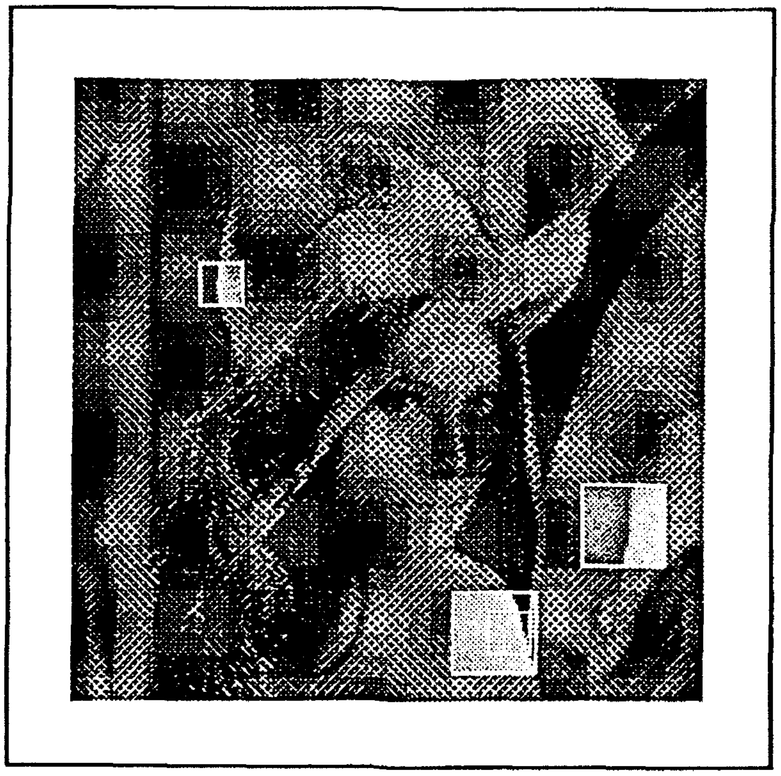


Figure 3-6 Self similar portions of the Lena image.

Experimental results suggest that most images that one would expect to encounter can be compressed by taking advantage of this type of self-similarity; for example, images of trees, faces, houses, mountains, clouds, etc.

4.0 Fractal Data Compression

4.1 Approach

Our approach to fractal data compression is to partition the image. This may be described in terms of the copying machine metaphor introduced in the background section. The partitioned copy machine has four variable components:

- the number of copies of the original pasted together to form the output,
- a setting of position and scaling, stretching, skewing and rotation factors for each copy.

These features are a part of the copying machine that can be used to generate the images in Figure 3-3. We add to the following two capabilities:

- a contrast and brightness adjustment for each copy,
- a mask which selects, for each copy, a part of the original to be copied.

These extra variables are sufficient to allow the encoding of grey-scale images. The last partitions an image into pieces which are each transformed separately. By partitioning the image into pieces, we allow the encoding of many shapes that are difficult to encode using an IFS.

Let us review what happens when we copy an original image using this machine. Each lens selects a portion of the original, which we denote by D_i and copies that part (with a brightness and contrast transformation) to a part of the produced copy which is denoted R_i. We call D_i domains and the R_i ranges. We denote this transformation by w_i. The partitioning is implicit in the notation, so that we can use almost the same notation as with an IFS. Given an image f, one copying step in a machine with N lenses can be written as $W(f) = w_1(f) \cup w_2(f) \cup \ldots \cup w_N(f)$. As before the machine runs in a feedback loop; its own output is fed back as its new input again and again.

We call the mathematical analogue of a partitioned copying machine, a partitioned iterated function system (PIFS). The grey level adds another dimension, so the transformations w_i are of the form,

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix} \qquad (4\text{-}1)$$

where s_i controls the contrast and o_i the brightness of the transformation.

It is convenient to write

$$v_i(x,y) = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix} \tag{4-2}$$

Since an image is modeled as a function $f(x,y)$, we can apply w_i to an image f by $w_i(f) \equiv w_i\ (x, y, f\ (x,y\ ))$. Then v_i determines how the partitioned domains of an original are mapped to the copy, while s_i and o_i determine the contrast and brightness of the transformation. It is always implicit, and important to remember, that each w_i is restricted to $D_i \times I$, the vertical space above D_i. That is w_i applies only to the part of the image that is above the domain D_i. This means that $v_i(D_i) = R_i$.

Since we want $W(f)$ to be an image, we must insist that $\cup R_i = I^2$ and that $R_i \cap R_j = \varnothing$ when $i \neq j$. That is, when we apply W to an image, we get some single valued function above each point of the square I^2. Running the copying machine in a loop means iterating the map W. We begin with an initial image of and then iterate $f_1 = W(f_0), f_2 = W(f_1) = W(W(f_0))$, and so on. We denote the *n-th* iterate by $f_n = W^{\circ n}(f_0)$.

Fixed Points for PIFS

In our case, a fixed point is an image f that satisfies $W(f) = f$; that is, when we apply the transformation to the image, we get back the original image. The contractive mapping theorem says that the fixed point of W will be the image we get when we compute the sequence $W(f_0), W(W(f_0)), W(W(W(f_0))), \ldots$, where f_0 is *any* image. So if we can be assured that W is contractive in the space of all images, then it will have a unique fixed point which will then be some image.

Because the metric we chose in equation 1 is only sensitive to what happens in the z direction, it is not necessary to impose contractivity conditions in the x or y directions. The transformation W will be contractive when each $s_i < 1$; that is, when z distances are shrunk by a factor less than 1. In fact, the contractive mapping principle can be applied to $W^{\circ m}$ (for some m), so it is sufficient for $W^{\circ m}$ to be contractive. This leads to the somewhat surprising result that there is no specific condition on any specific s_i either. In practice, it is safest to take $s_i < 1$ to ensure contractivity. but we know from experiments that taking $s_i < 1.2$ is safe and that this results in slightly better encodings.

Eventually Contractive Maps

When W is not contractive and $W^{\circ m}$ is contractive, we call W *eventually contractive.* A brief explanation of how a transformation W can be eventually contractive but not contractive is in order. The map W is composed of a union of maps w_i operating on disjoint parts of an image. The iterated transform $W^{\circ m}$ is composed of a union of compositions of the form

$$w_{i_1} \circ w_{i_2} \circ \ldots w_{i_m} \tag{4-3}$$

It is a fact that the product of the contractivities bounds the contractivity of the compositions, so the compositions will be contractive if each contains sufficiently contractive w_{i_j}. Thus W will be eventually contractive (in the sup metric) if it constrains sufficient "mixing " so that the contractive w_i eventually dominate the expansive ones. In practice, given a PIFS this condition is simple to check in the sup metric.

Suppose that we set all the $s_i < 1$. This means that when the copying machine is run, the contrast is always reduced. This seems to suggest that when the machine is run in a feedback loop, the resulting attractor will be an insipid, contrast-less grey. This is not so because contrast is created between ranges which have different brightness levels o_i. If we take the v_i to be contractive, then the places where there is contrast between the R_i in the image will propagate to smaller and smaller scale, and this is how detail is created in the attractor. This is one reason to require that the v_i be contractive.

We now know how to decode an image that is encoded as a PIFS. Start with any initial image and repeatedly run the copy machine, or repeatedly apply W until we get close to the fixed point f_∞. We will use Hutchinson's notation and denote this fixed point by $f_\infty = |W|$. The decoding is easy, but it is the encoding which is interesting. To encode an image we need to figure out R_i, D_i and w_i, as well as N, the number of maps w_i we wish to use.

Encoding Images

Suppose we are given an image f that we wish to encode. This means we want to find a collection of maps $w_1, w_2 \ldots, w_n$ with $W = \bigcup_{i=1}^{N} w_i$ and $f = |W|$. That is, we want f to be the fixed point of the map W. The fixed point equation

$$f = W(f) = w_1(f) \cup w_2(f) \cup \ldots w_N(f) \tag{4-4}$$

suggests how this may be achieved. We seek a partition of f into pieces to which we apply the transforms w_i and get back f. This is too much to hope for in general, since images are not composed of pieces that can be transformed non-trivially to fit exactly somewhere else in the image. What we can hope to find is another image $f' = |W|$ *with* $\partial(f',f)$ small. That is we seek a transformation W whose fixed point $f' = |W|$ is close to, or looks like, f. In that case,

$$f \approx f' = W(f') \approx W(f) = w_1(f) \cup w_2(f) \cup \ldots w_N(f). \tag{4-5}$$

Thus it is sufficient to approximate the parts of the image with transformed pieces. We do this by minimizing the following quantities

$$\partial(f \cap (R_i \times I), w_i(f)) \quad i = 1,\ldots,N \tag{4-6}$$

That is, we find pieces D_i and maps w_i, so that when we apply a w_i to the part of the image over D_i, we get something that is very close to the part of the image over R_i. Finding the pieces R_i (and corresponding D_i) is the heart of the problem.

Least Squares

Given two squares containing n pixel intensities, $a_1,\ldots,a_n$ (from D_i) and $b_1,\ldots,b_n$ (from R_i). We can seek s and o to minimize the quantity

$$R = \sum_{i=1}^{n} (s \cdot a_i + o - b_i)^2 \tag{4-7}$$

This will give us a contrast and brightness setting that makes the affinely transformed a_i values have the least squared distance from the b_i values. The minimum of R occurs when the partial derivatives with respect to s and o are zero, which occurs when

$$s = \frac{\left[n^2\left(\sum_{i=1}^{n} a_i b_i\right) - \left(\sum_{i=1}^{n} a_i\right)\left(\sum_{i=1}^{n} b_i\right)\right]}{\left[n^2 \sum_{i=1}^{n} a_i^2 - \left(\sum_{i=1}^{n} a_i\right)^2\right]} \tag{4-8}$$

and

$$o = \frac{\left[\sum_{i=1}^{n} b_i - s \sum_{i=1}^{n} a_i\right]}{n^2} \tag{4-9}$$

In that case,

$$R = \frac{\left[\sum_{i=1}^{n} b_i^2 + s(s \sum_{i=1}^{n} a_i^2 - 2(\sum_{i=1}^{n} a_i b_i) + 2o \sum_{i=1}^{n} a_i) + o(on^2 - 2 \sum_{i=1}^{n} b_i)\right]}{n^2} \tag{4-10}$$

$$\text{If } n^2 \sum_{i=1}^{n} a_i^2 - (\sum_{i=1}^{n} a_i)^2 = 0,$$
$$\text{then } s = 0 \text{ and } o = \sum_{i=1}^{n} \frac{b_i}{n^2} \tag{4-11}$$

4.3 Accomplishments

During our Phase II effort we explored several topics relating to the improvement of our image encoding and decoding of algorithms.

4.3.1 Image Encoding

Our efforts in the area of image encoding have been directed toward encoding optimization i.e., reducing the number of correlations which must be computed, more effective tiling schemes and color encoding. We also investigated optical and Fourier transform methods in cooperation with a research group at Foster-Miller.

Quadtree Partitioning

The original reason for developing quadtree partitioning was that in general there are regions of images that are difficult to cover well by tiles of fixed size (for example, Lena's eyes). Similarly, there are regions that could be covered well with larger R_i, thus reducing the total number of w_i maps needed (and increasing the compression of the image). Thus quadtree is a generalization of the fixed size R_i. In a quadtree partition, a square in the image is broken up into 4 equally size sub-squares, when it is not covered well enough by a domain. This process repeats recursively starting from the whole image and continuing until the squares are small enough to be covered within some specified rms tolerance. Small squares can be covered better than large ones because contiguous pixels in an image tend to be highly correlated.

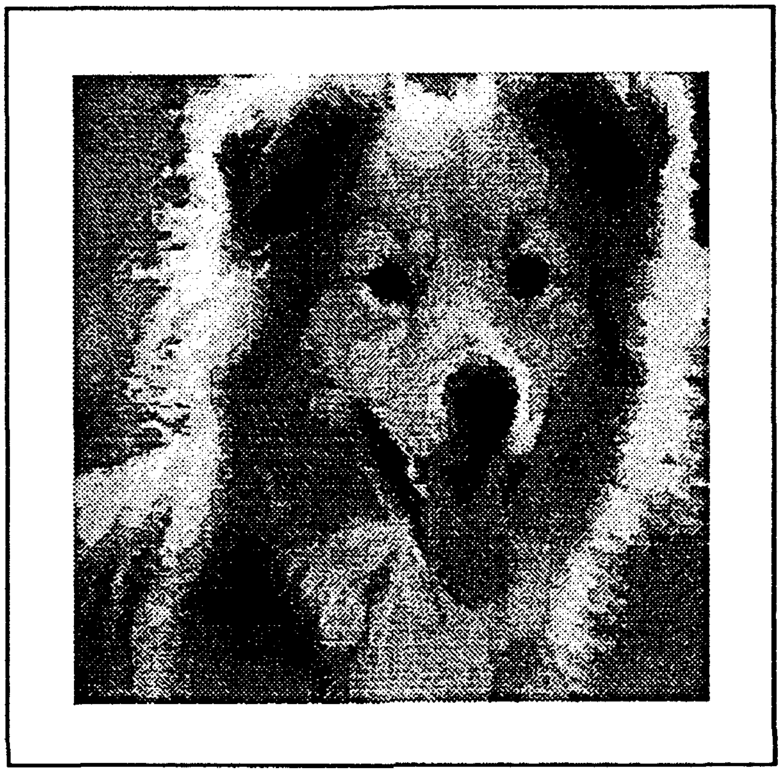


Figure 4-1 A collie (256 x 256) compressed with the quadtree scheme at 28.95:1 with an rms error of 8.5. (From Phase I effort)

An algorithm that works well for encoding 256 x 256 pixel images based on this idea can proceed as follows (see [9]). Choose for the collection $\boldsymbol{D}$ of permissible domains all the sub-squares in the image of size 8, 12, 16, 24, 32, 48 and 64. Partition the image recursively by a quadtree method until the squares are of size 32. For each square in the quadtree partition, attempt to cover it by a domain that is larger; this makes the v_i contractive. If a predetermined tolerance rms value e_c is met, then call the square R_i and the covering domain D_i. If not, then subdivide the square and repeat. This algorithm works well. It works even better if diagonally oriented squares are used in the domain pool $\boldsymbol{D}$ also. Figure 4-1 shows an image of a collie compressed using the scheme.

HV-Partitioning

One weakness of the quadtree based partitioning is that is makes no attempt to select the domain pool $\boldsymbol{D}$ in a content dependent way. The collection must be chosen to be very large so that a good fit to a given range can be found. A way to remedy this, while increasing the flexibility of the range partition, is to use an HV-partition. In an HV-partition, a rectangular image is recursively partitioned either horizontally o1 vertically to form two new rectangles. The partitioning repeats recursively until a covering tolerance is satisfied, as in the quadtree scheme.

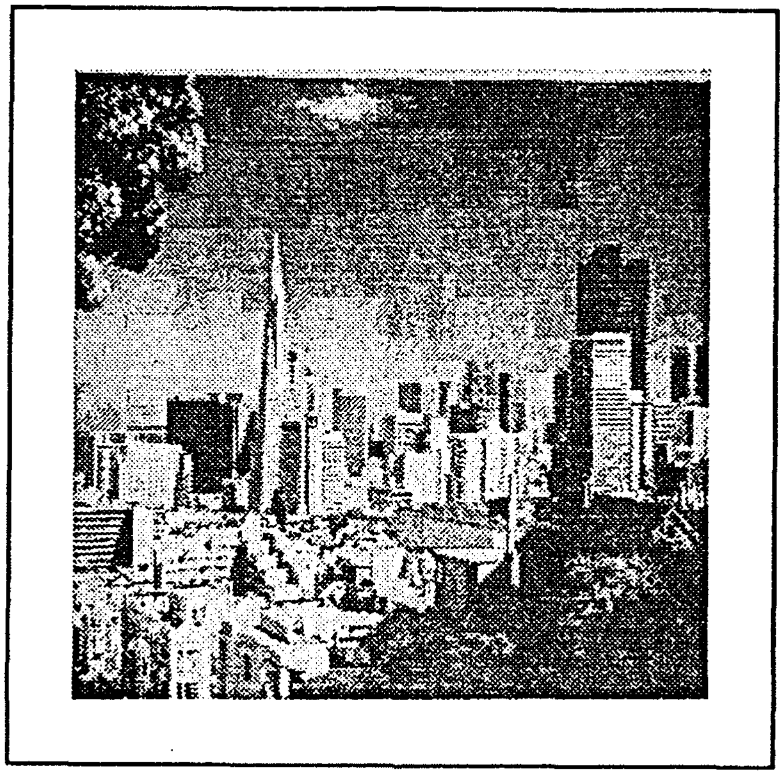


Figure 4-2 San Francisco (256 x 256)compressed with the HV scheme at 7.6:1 with an rms error of 7.1.

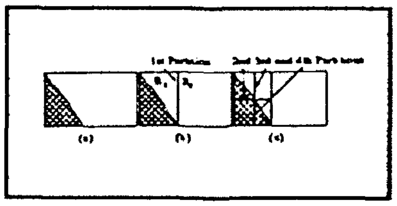


Figure 4-3 The HV scheme attempts to create self similar rectangles at different scales.

This scheme is more flexible, since the position of the partition is variable. We can then try to make the partitions in such a way that they share some self similar structure. For example, we can try to arrange the partitions so that edges in the image will tend to run diagonally through them. Then, it is possible to use the larger partitions to cover the smaller partitions with a reasonable expectation of a good cover. Figure 4-3 demonstrates this idea. The figure shows a part of an image (a); in (b) the first partition generates two rectangles, R_1 with the edge running diagonally through it, and R_2 with no edge; and in (c) the next three partitions of R_1 partition it into 4 rectangles, two rectangles which can be well covered by R_1 (since they have an edge running diagonally) and two which can be covered by R_2 (since they contain no edge). Figure 4-2 shows an image of San Francisco encoded using this scheme.

Triangular Partitioning

A third way to partition an image is based on triangles. In the triangular partitioning scheme, a rectangular image is divided diagonally into two triangles. Each of these is recursively subdivided into 4 triangles by segmenting the triangle along lines that join three partitioning points along the three sides of the triangle. This scheme has several potential advantages over the HV-partitioning scheme. It is flexible, so that triangles in the scheme can be chosen to share self-similar properties, as before. The artifacts arising from the covering do not run horizontally and vertically, and this is less distracting. Also, the triangles can have any orientation, so we break away from the rigid 90 degree rotations of the quadtree and HV partitioning schemes. This scheme, however, remains to be fully developed and explored.

Figure 4-4 shows sample partitions arising from the three partitioning schemes applied to the Lena image.

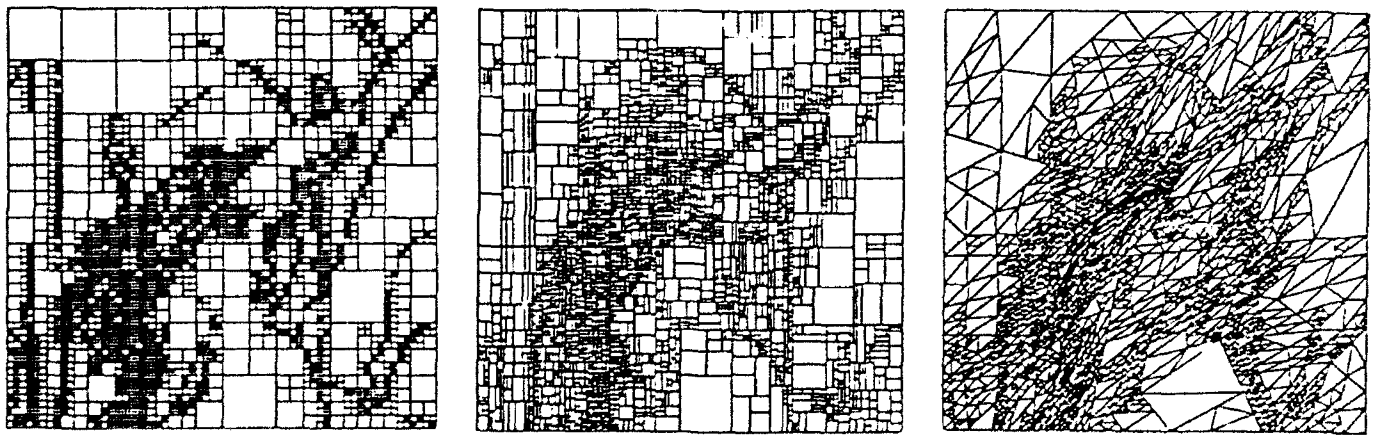


Figure 4-4 A quadtree partition (5008 squares), and HV partition (2910 rectangles), and a triangular partition (2954 triangles).

The pseudo-code in Table 4-1 shows two ways of encoding images using the idea presented. One method attempts to target a fidelity by finding a covering such that the quantities (see equ. 3-9, 4-6) are below some criterion e_c. The other method attempts to target a compression ratio by limiting the number of transforms used in the encoding.

Table 4-1 Two pseudo-codes for an adaptive encoding algorithm

```
• Choose a tolerance level $e_c$.
• Set $R_1 = I^2$ and mark it uncovered.
• While there are uncovered ranges $R_i$ do {
    • Out of the possible domains $\mathbf{D}$, find the domain $D_i$
      and the corresponding $w_i$ which best covers $R_i$ (i.e.
      which minimizes expression (4)).
    • If $\delta(f \cap (R_i \times I), w_i(f)) < e_c$ or size$(R_i) \leq r_{min}$ then
        • Mark $R_i$ as covered, and write out the transfor-
          mation $w_i$;
    • else
        • Partition $R_i$ into smaller ranges which are
          marked as uncovered, and remove $R_i$ from the list
          of uncovered ranges.
  }
```

a. Pseudo-code targeting a fidelity e_c.

```
• Choose a target number of ranges $N_r$.
• Set a list to contain $R_1 = I^2$, and mark it as uncovered.
• While there are uncovered ranges in the list do {
    • For each uncovered range in the list, find and store
      the domain $D_i \in \mathbf{D}$ and map $w_i$ which covers it best,
      and mark the range as covered.
    • Out of the list of ranges, find the range $R_j$ with
      size$(R_j) > r_{min}$ which has the largest

                 $\delta(f \cap (R_j \times I), w_j(f))$

      (i.e. which is covered worst).
    • If the number of ranges in the list is less than $N_r$
      then {
        • Partition $R_j$ into smaller ranges which are added
          to the list and marked as uncovered.
        • Remove $R_j, w_j$ and $D_j$ from the list.
      }
  }
• Write out all the $w_i$ in the list.
```

b. Pseudo-code targeting a compression having N transformations.

4.3.1.2 Storing the Encoding Compactly

To store the encoding compactly, we do not store all the coefficients in equation (4-1). The contrast and brightness settings are stored using a fixed number of bits. One could compute the optimal s_i and o_i and then discretize them for storage. However, a significant improvement in fidelity can be obtained if only discretized s_i and o_i values are used when computing the error during encoding. Using 5 bits to store s_i and 7 bits to store o_i has been found empirically optimal in general. The distribution of s_i and o_i shows some structure, so further compression can be attained by using entropy encoding.

The remaining coefficients are computed when the image is decoded. In their place we store R_i and D_i. In the case of a quadtree partition, R_i can be encoded by the storage order of the transformations if we know the size of R_i. The domains D_i must be stored as a position and size (and orientation if diagonal domains are used). This is not sufficient, though since there are 8 ways to map the four corners of D_i to the corners of R_i. So we also must use 3 bits to determine this rotation and flip information.

In the case of the HV-partitioning and triangular partitioning, the partitions stored as a collection of offset values. As the rectangles (or triangles) become smaller in the partition, fewer bits are required to store the offset value. The partition can be completely reconstructed by the decoding routine. One bit must be used to determine if a partition is further subdivided or will be used as an R_i and a variable number of bits must be used to specify the index of each D_i in a list of all the partitions. For all three methods, and without too much effort, it is possible to achieve a compression of roughly 31 bits per w_i on average.

The partitioning algorithms described are adaptive in the sense that they utilize a range size which varies depending on the local image complexity. For a fixed image, more transformations lead to better fidelity but worse compression. This trade-off between compression and fidelity leads to two different approaches to encoding an image f - one targeting fidelity and one targeting compression. These approaches are outlined in the pseudo-code in table 4-1. In the table, size (R_i) refers to the size of the range; in the case of rectangles, size (R_i) is the length of the longest side.

The images in Appendix 2 show typical results for our HV partitioning scheme.

4.3.1.3 Improving Encoding Speed

Our encoding algorithm consists of searching through a large number of sub-images, in order to find one which has a low RMS error when correlated with a piece of a quadtree partition of the image. In order to reduce this search time, we classify the collections of sub-images. The search is limited to elements of the same class, leading to a reduced search time.

Our classification is as follows:

> Each sub-image is first partitioned into 4 equal quadrants, which are ordered by brightness. This ordering leads to 3 classes, once a rotation and a reflection are used to bring the brightest quadrant into the upper left position. This gives three classes and also determines a rotation (reducing the search class by a factor of 4 or 8, the number of symmetry operations on the square or rectangle respectively.
>
> These three classes are further split into 12 or 24 by ordering the contrast levels of the sub-quadrants of each quadrant. This gives a total of 36 or 72 classes.

We were able to achieve a 15-fold increase in encoding speeds. Typical times for an Intel 80486-33 are given in Table 4-2. Phase I time values are approximate due to a change in the development platform.

4.3.1.4 Optical Methods

In anticipation of future developments in optical processing, we investigated the feasibility of performing tile comparisons optically. This was carried out in collaboration with Foster-Miller. The method we investigated was optical four wave mixing.

Optical four wave mixing can be used to construct the convolution correlation of four images. This operation may be represented by

$$I_r = I_1 \bullet (I_2 * I_3), \qquad (4\text{-}12)$$

where I_r, I_1, I_2, and I_3 denote the output and the three input images respectively, while $\bullet$ represents the correlation operation and $*$ represents the convolution operation.

The major bottleneck in computing an IFS code for an image is checking the large set of possible linear transforms and windows. In particular, it is necessary to compute

$$I_w = T\,(W \cdot I) \bullet I, \tag{4-13}$$

for a large number of linear transforms, T, and windows, W. In this formula, $\cdot$ represents multiplication while $\bullet$ represents correlation. In particular $W \cdot I$ is the windowed image corresponding to a tile R_i in some partition above.

In order to put this into the form of a three wave product we use the convolution theorem:

$$W \cdot I = F^{-1}\,(FW * FI), \tag{4-14}$$

where F and F^{-1} correspond to the Fourier transform and inverse Fourier transform respectively. Applying Formula (4-14) to Formula (4-13) we obtain

$$I_w = TF^{-1}\,(FW * FI) \bullet I. \tag{4-15}$$

The composition of operators TF^{-1} can also be written $F^{-1}\,T_F$, where T_F is the linear transform induced by T in the spatial frequency domain.

The operation described by Equation (4-15) was implemented by means of optical techniques by a group under Larry Domash of Foster Miller [26] and [27]. Figure 4-5, supplied by Dr. Domash's group, shows a test case based on the Spierpinsky triangle.

We also modeled the optical process described above by means of the FFT. To do this, we developed an FFT code for comparing domain and range tiles. The FFT replaces a large portion of the innermost loop of our present code. In essence, the code replaces repeated computation of cross correlations by a single computation, based on the convolution property of the Fourier transform. By taking advantage of the fact that one of the inputs to the convolution computation is mostly zero and the fact that the FFT of the image need only be computed once, we can reduce the convolution to one FFT per domain tile. Our benchmark studies indicated that optical methods would give a considerable speedup of the compression algorithm, but unfortunately optical methods do not presently yield the accuracies we require.

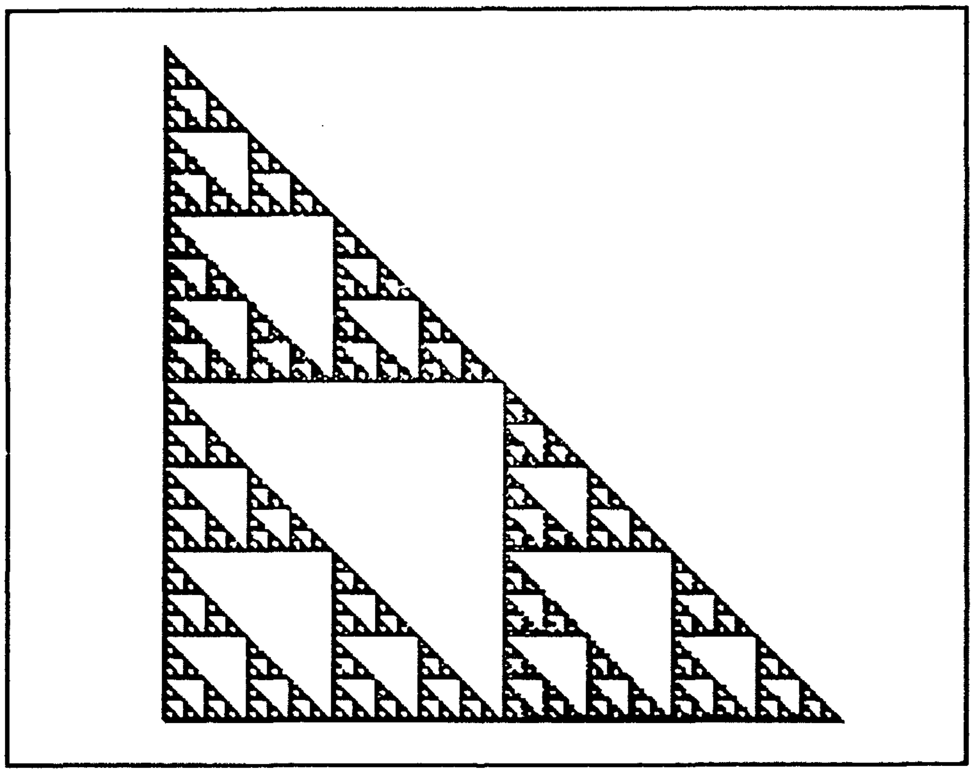
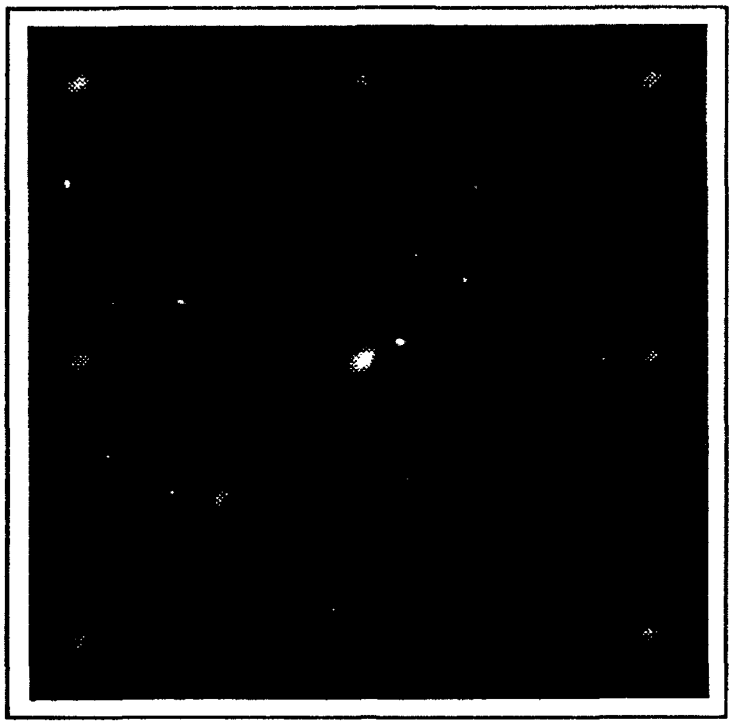


Figure 4-5 256 x 256 Spierpinski triangle.
The autocorrelation of the test image was convolved with 4 x 4, 8 x 8, and 16 x 16 pixel squares. The results are 8-bit (256 levels) 256 x 256 images presented in grayscale form in figures 4-6 to 4-7.


a) Unsaturated

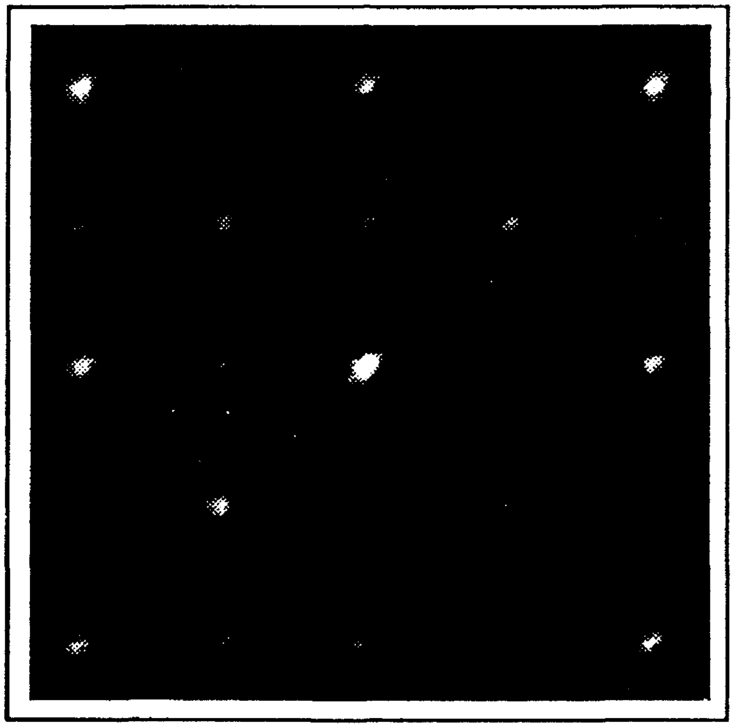
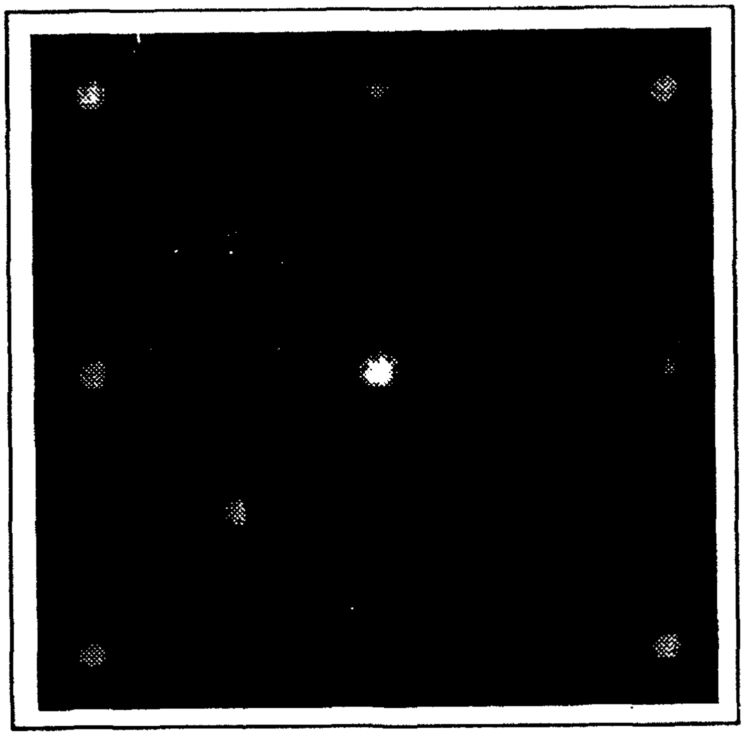
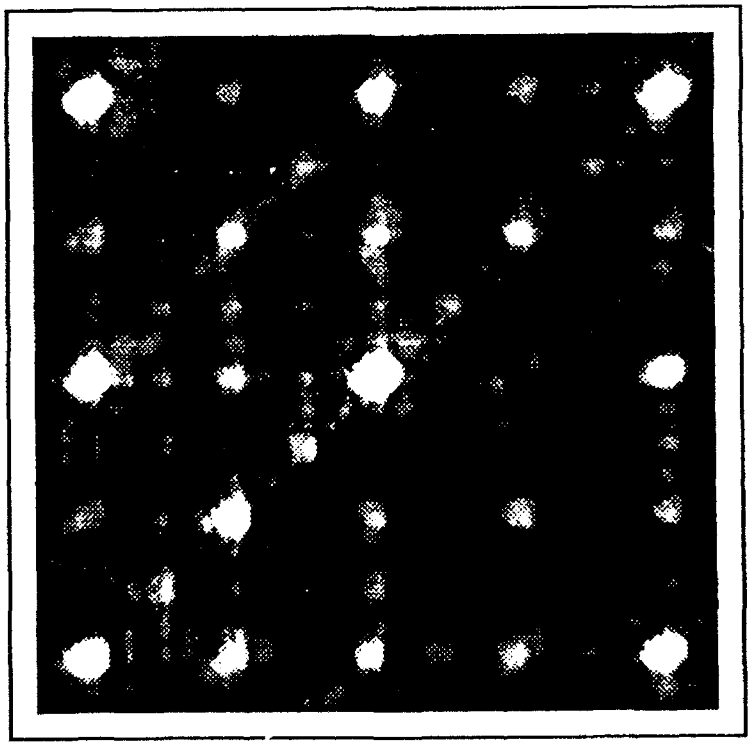

b) Saturated

Figure 4-6 Grayscale optical result produced by convolution with a 4 x 4 square. (Supplied by Foster-Miller)



a) Unsaturated



b) Saturated

Figure 4-7 Grayscale optical result produced by convolution with a 8 x 8 square. (Supplied by Foster-Miller)

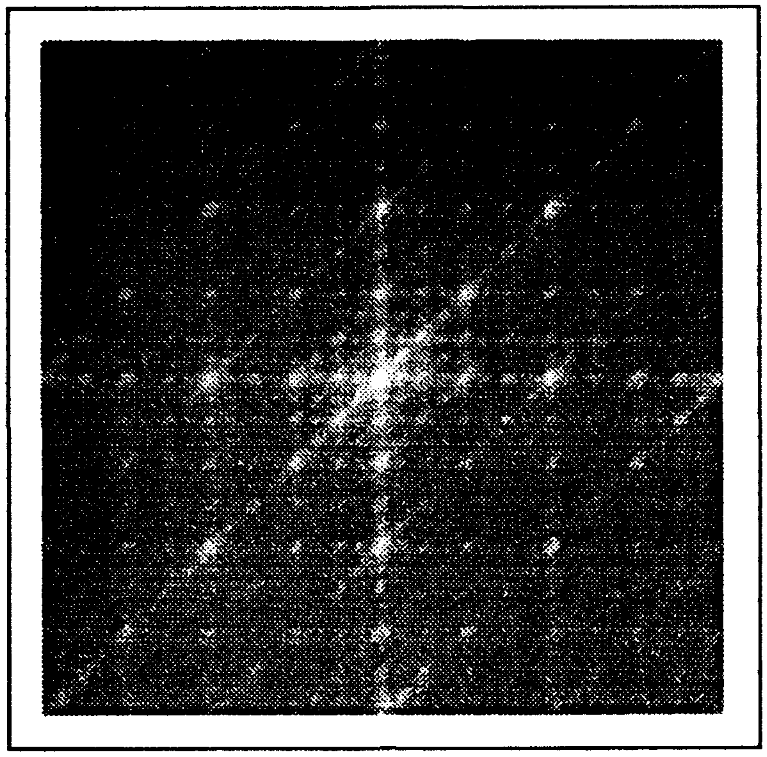


Figure 4-8 Computer simulation of convolution - correlation of the Spierpinski triangle with the 4 x 4 square. (Supplied by Foster-Miller)

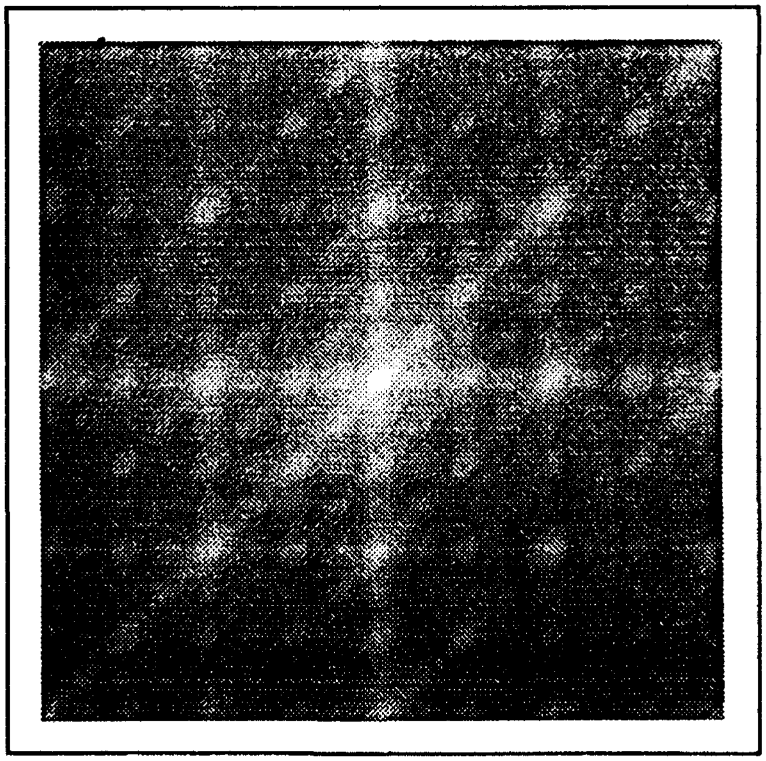


Figure 4-9 Computer simulation-correlation with 8 x 8 square. (Supplied by Foster-Miller)

4.3.1.5 Color Image Encoding

We have developed several methods of color encoding. Typically, color images are stored as indices to a color look-up table. This means that an 8 bit per pixel image can contain 256 colors from a palette which contains typically 2^{24} colors. Encoding such images is done usually by expanding the data to a red-green-blue (RGB) representation, converting this representation to a YIQ representation (also used in color television transmission) and encoding the three signals separately. The advantage of this scheme is that the I and Q signals can be stored very compactly with very little perceptual degradation. The disadvantage is that three images must be encoded.

The implementation of our algorithm involves some tricky color look-up table manipulation because screen displays have a palette of 256 colors, while the YIQ signals typically yield a much larger number of colors. To circumvent the problem, we store the color map of the original image along with the compressed image. The display process finds the color from the table that is closest to the YIQ color from the decompression. Our algorithms presently handle 8-bit RGB color map or 24-bit RGB .SUN files, and 8-bit gray-scale Red, Green, and Blue components.

4.3.2 Image Decompression

Rapid image decompression is crucial for many applications of the fractal image compression scheme. During the course of our research we investigated both iterative and Gaussian pivoting schemes for decoding images.

Iterative Schemes

When decoding during the iterative method, an initial image $f_{\{0\}}$ in L^2 on $[0,1] \times [0,1]$ is used to compute the iterates $f_{\{n\}} = W(f_{\{n-1\}})$, where

$$W(\cdot) = \bigcup_i W_i(\cdot) \tag{4-16}$$

This can also be written as

$$f_{\{n\}}(x,y) = \sum_i a_i f_{\{n-1\}}\left(V_i^{-1}(x,y)\right) + o_i \tag{4-17}$$

where the i^{th} term of the summation is defined on the region R_i, V_i is an affine transformation on the region R_i, and a_i and o_i are contrast and intensity adjustments respectively.

We implemented integer versions of the iteration scheme with a variant on the storage of the transforms. In the iterative scheme, each pixel depends upon another (with a brightness and a contrast adjustment). Rather than computing this dependency from the transformations at each iteration step, we compute it initially. This also appears to lead to considerable improvement in decoding times. The final version of our algorithm handles color, smoothing to eliminate artifacts, and sampling or averaging of domain pixels onto the range pixels.

Gaussian Pivoting

Suppose we are decompressing an image of $M \times M$ pixels. We can write the image as a column vector, and then the above equation can be written.

$$f_{\{n\}} = \mathrm{S} f_{\{n-1\}} + O \qquad (4\text{-}18)$$

where S is an $M^2 \times M^2$ matrix with entries s_{ij} which encode the contrast adjustments a_i and spatial affine transformations V_i and O is a column vector encoding the intensity adjustments. Then

$$f_{\{n\}} = \mathrm{S}^n f_{\{0\}} + \sum_{i}^{n-1} \mathrm{S}^i O \qquad (4\text{-}19)$$

If each s_{ij} is less than 1 then the first term is 0 in the limit. This condition can be relaxed if W is eventually contractive. When I - S is invertible

$$f_\infty = \sum_{i}^{\infty} \mathrm{S}^i O = (I - \mathrm{S})^{-1} O \qquad (4\text{-}20)$$

If each pixel value of $f_{\{n\}}$ depends upon only one (or even a few) pixel values of $f_{\{n-1\}}$ then I - S is very sparse and is inverted readily by sparse matrix methods. We have implemented this algorithm and it shows promising results. It runs considerably faster than the previous version of the iteration algorithm.

We developed a decoding algorithm which triangularizes the S matrix by pivoting. We have observed that different pivoting schemes lead to different decoding times since each pivot operation eliminates an entry in one place in the matrix and creates another somewhere else. If the new entry is located above the diagonal, no further pivot is required, but if it is located below, another pivot will be required. We studied de referencing schemes which allow several simultaneous pivot operations in the case where one pivot requires another, but were not able to improve the Gaussian pivoting scheme in speed sufficiently to warrant using pivoting instead of the iteration scheme.

Image	*Lena 256 x 256*	*Lena 512 x 512*		
Phase I				
S/N Ratic (dB)	28.8	29.2	30.0	32.1
Compression Ratio	11.3	38.7	23.5	15.6
Compression Time (min)	19.6	-----	215.2	75.0
Decompression Time (sec)	13	52	52	52
Phase II				
S/N Ratio	28.7	29.3	30.4	32.1
Compression Ratio	19.4	55.6	40.8	26.7
Compression Time (min) (486/33)	2.40	17.03	15.13	26.35
Decompression Time (sec) (486/33)	2.25	12.03	12.30	12.69

Table 4-2 Comparison of Phase I and Phase II Results

4.3.2.1 Number of Decompression Passes

Independent of the encoding process, we can affect the quality of the reconstructed image by the number of decoding passes we select. See Table 4-3.

Image	*#Passes*	*Decompression Time*	*S/N Ratio*	*Compression Ratio*
Lena 256 x 256	1	0.60	27.62	19.418
Lena 256 x 256	2	1.15	28.67	19.418
Lena 256 x 256	3	1.70	28.72	19.418
Lena 256 x 256	4	2.25	28.73	19.418
Lena 256 x 256	5	2.80	28.73	19.418
Lena 512 x 512	1	3.51	28.88	60.249
Lena 512 x 512	2	6.38	29.15	60.249
Lena 512 x 512	3	9.23	29.16	60.249
Lena 512 x 512	4	12.03	29.16	60.249
Lena 512 x 512	5	14.88	29.16	60.249
Lena 512 x 512	1	3.68	31.60	27.014
Lena 512 x 512	2	6.64	32.26	27.014
Lena 512 x 512	3	9.67	32.30	27.014
Lena 512 x 512	4	12.68	32.30	27.014
Lena 512 x 512	5	15.71	32.30	27.014

Table 4-3 Ratios vs. Passes

5.0 Wavelet Image Compression

5.1 Background

Because the fractal compression method has not been entirely satisfactory for some types of images such as maps, fingerprints and satellite images we have implemented a version of the wavelet compression method reported by Antonini [1].

This encoding method has three main steps: a wavelet transform followed by a lattice vector quantization followed by a Huffman encoding of the output vectors. The Huffman code step is not reported in the report by Antonini [1]. A standard codebook may be used for transmission, with a provision for sending codebook entries for rarely-encountered vectors.

5.1.1 One Dimensional Wavelets

Wavelets are functions generated from one single function ψ by deletions and translations

$$\psi_{a,b}(x) = |a|^{-1/2} \psi\left(\frac{x-b}{a}\right) \tag{5-1}$$

Generally we require that the metric wavelet ψ should be well localized in time and frequency and have zero mean.

The basic idea of the wavelet transform is to represent any arbitrary function f as a superposition of wavelets. For example we may write

$$f(x) = \iint dadb\ c(a,b)\ \psi_{a,b}(x) \tag{5-2}$$

In practice we deal with discrete decompositions

$$f = \sum_{m,n} c_{m,n} \tilde{\psi}_{a,b} \tag{5-3}$$

Where

$$\psi_{m,n}(x) = 2^{-m/2} \tilde{\psi}\left(2^{-m}x - n\right) \tag{5-4}$$

is the dual basis to $\psi_{m,n}$ and

$$c_{m,n} = (\psi_{m,n}, f) = \int \psi_{m,n}(x) f(x)\, dx \tag{5-5}$$

The basic wavelet is constructed from a scaling function φ such that

$$\varphi(x) = \sum_k \alpha_k \varphi(2x - k) \tag{5-6}$$

such that

$$\int \varphi(x)\, dx = 1 \tag{5-7}$$

and $\psi(x)$ is given by

$$\psi(x) = \sum_k (-1)^k \alpha_{k+1} \varphi(2x + k) \tag{5-8}$$

The quantities α_k are the wavelet coefficients. In order to measure orthagonality we require that

$$\begin{aligned} &\sum_k \alpha_{2k+i} = 2^{-1/2} \quad i = 1,2 \\ &\sum_k \alpha_k \alpha_{k+2l} = \begin{cases} 2 \text{ if } l = 0 \\ 0 \text{ if } l \neq 0 \end{cases} \end{aligned} \tag{5-9}$$

5.1.2 Quadrature Mirror Filters

The wavelet coefficients define a low pass filter sequence. Following the notation of Beylkin, et al [4] we write the conditions

(1) $$\sum_j |h_j|\, |j|^{\epsilon} < \infty \quad \text{for some } \epsilon \tag{5-10}$$

(2) $$\sum_j h_{2j+i} = 2^{-1/2} \quad i = 0, 1 \tag{5-11}$$

(3) $$\sum_j h_j h_{j+2k} = \delta_k \qquad (5\text{-}12)$$

Note that the h's replace the α's above. Let $g = \{g\}$ be defined by $g_j = (-1)^j h_{1-j}$. Then $\{h,g\}$ is a pair of quadrature mirror filters and we can define two operations H and G. For any function f,

$$Hf(x) = \sum_j h_j f(2x - j) \qquad (5\text{-}13)$$

$$Gf(x) = \sum_j g_j f(2x - j) \qquad (5\text{-}14)$$

If the sequences g, h are finite we can define $\varphi = \lim_{n\to\infty} H^n(x)$ where X is the indicator function of [-1/2,1/2]. Note that φ is the unique fixed function of the equation $\varphi = H\varphi$.

Furthermore G and H have adjoints

$$H^*f(x) = \frac{1}{2}\sum_j h_j f\left(\frac{x}{2} + \frac{j}{2}\right) \qquad (5\text{-}15)$$

$$G^*f(x) = \frac{1}{2}\sum_j g_j f\left(\frac{x}{2} + \frac{j}{2}\right) \qquad (5\text{-}16)$$

We have the identities $HH^x=GG^*=I$, $HG^*=GH^*= 0$ and $H^*H+G^*G=I$. These identities can be used in the construction of multiresolution pyramids (as in Mallat [20] or Beylkin, et al [4]).

5.1.3 Multidimensional Analysis

Our research is based on the non-separable and non-oriented filters described by Antonini, et al [1]. We decompose images with a multiresolution scale factor $\sqrt{2}$. The scaling function is defined via

$$\phi_{m,n} = 2^m \, \phi\,(L^{2m}(x,y) - n) \quad n = (n_x, n_y) \qquad (5\text{-}17)$$

This translates into a pair of filters as given in Table 5-1.

```
            a
         f  b  f
      j  g  c  g  j
   f  g  i  d  i  g  f
a  b  c  d  e  d  c  b  a
   f  g  i  d  i  g  f
      j  g  c  g  j
         f  b  f
            a
```

	h	$\bar{h}$
a	+0.001671	—
b	-0.002108	-0.005704
c	-0.019555	-0.007192
d	+0.139756	+0.164931
e	+0.687859	+0.586315
f	+0.006687	—
g	-0.006324	-0.017113
i	-0.052486	-0.014365
j	+0.010030	—

Table 5-1 Wavelet Filter Pairs

5.1.4 Vector Quantization

Following the calculation of the wavelet coefficients we employed a vector quantization procedure based on a regular lattice. The remainder of this section is quoted from Antonini, et al [1].

An n-dimensional lattice λ_n is defined as a set of vectors

$$\lambda_n = (X/X = u_1 a_1 + ... + u_n a_n)$$

where $a_1, ..., a_n$ are linearly independent vectors in m-dimensional real Euclidian space $\mathbb{R}^m$ with $m \geq n$, and $u_1, ..., u_n$ are integers.

Some important n-dimensional lattices are the root lattices A_n $(n\geq 1)$, D_n $(n\geq 2)$, and E_n $(n=6,7,8)$, and the Barnes-Wall lattice Λ_{16}. These lattices give the best sphere packings and coverings [9]. For our application, we have used the Λ_{16} lattice.

For $n\geq 2$ the D_n lattice is defined as follows

$$D_n = \{(x_1, x_2, ..., x_n) \in \mathbb{Z}^n / x_1 + x_2 + ... + x_n \ even\}$$

and consists of those points of the $\mathbb{Z}^n$ rectangular lattice whose coordinate sum is even.

The E_8 and Λ_{16} lattices are constructed using respectively the D_8 and D_{16} lattices.

The E_8 lattice is defined by

$$E_8 = D_8 \cup \left(\left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2} \right) + D_8 \right)$$

E_8 consists of the points $(x_1, \ldots, x_8)$ with $x_i \in \mathbb{Z}$ and $\sum_i x_i$ even, together with the points $(y_1, \ldots y_8)$ with $y_i + 1/2 \in \mathbb{Z} + 1/2$ and $\sum_i y_i$ even.

Finally, the Barnes-Wall lattice is constructed by

$$\Lambda_{16} = \cup_{i=0}^{32} \left(r_i + 2D_{16} \right)$$

where the translate vectors r_i correspond to the lines (or columns) of an Hadamard matrix $\tilde{H}_{16}$ with $(-1,1) \rightarrow (1,0)$, and to the lines (or columns) of $\tilde{\bar{H}}_{16}$ the complemented matrix.

If the lattice is used as a quantizer, all the points in the Voronoi region around the lattice point X are represented by X in terms of mean squared error (MSE) distortion. In fact, a lattice quantizer may be defined as a quantizer whose output set $\mathcal{Y}$ is a subset of a lattice.

Conway and Sloane [8] have developed last quantization algorithms for the vector quantizers based upon the root lattice D_n. For a real number x_i let us define

$f(x_i)$ = closest integer to x_i and $w(x_i) = f(x_i) + \text{sign}\,(x_i - f(x_i))$

with $\text{sign}(\alpha) = 1 \quad \text{if } \alpha \geq 0$

$= -1 \quad \text{if } \alpha < 0$

Finding the closest point of D_n can be done by using the following procedure [8] :

- Given $X \in \mathbb{R}^n$
- Compute $f(X) = (f(x_1), \ldots, f(x_n))$ the closest point of $\mathbb{Z}^n$ to X.
- Compute $g(X) = (f(x_1), \ldots, w(x_i), \ldots, f(x_n))$ if x_i is the worst component of X : that furthest from an integer.
- The closest point of D_n is whichever of $f(X)$ and $g(X)$ have an even sum of components; in fact, one will have an even sum and the other an odd sum.

A procedure ϕ, for finding the closest point of a lattice λ to a given point X, can be easily converted to a procedure for finding the closest point of a coset $r+\lambda$ or a union of cosets $\cup_i(r_i + \lambda)$. In fact, if $\phi(X)$ is the closest point of λ to X, then $g = \phi(X-r) + r$ is the closest point of $r+\lambda$ to X. For a union of cosets we must compute all the $g_i = \phi(X-r_i) + r_i$ and choose the closest to X in terms of MSE distortion [8].

Since E_8 is the union of two cosets of D_8, and Λ_{16} is the union of 32 cosets of D_{16}, then we can use the previous encoding procedure to find the closest point of E_8 or Λ_{16} to X.

5.1.5 Huffman Encoding

We implemented a Huffman coding algorithm to compress the quantized coefficients of the wavelet decomposition. This well-known scheme assigns short code words to the most probable input symbols and longer code words to the least probable ones to obtain an encoding with a small average code length.

The input symbols to the Huffman encoder are quantized wavelet coefficients within 4 x 4 blocks, i.e. vectors of 16 elements, in the coefficient domain. The procedure below converts a list of the frequencies for each unique input symbol into a binary tree which provides the code for each of the input symbols. Each element in the frequencies list represents either a unique symbol or the sum of the frequencies of a subset of the unique symbols.

1: Search the list of frequencies for the two elements with the lowest frequencies.

2: Replace these two elements with their sum. The updated list has one less element. Both of the replaced elements will have Huffman codes one bit longer than their sum will have.

3: Go back to step 1 and repeat the procedure until the list contains only one element.

The external nodes of the tree represent the different vectors V1, V2, V3, ... We interpret a '0' as a left branch and a '1' as a right branch and get the Huffman codes for the vectors V1, V2, V3, ... respectively.

A concise description of the Huffman coding algorithm can be found in [15] and [23].

In our system, we limit the length of any Huffman code to 15 bits since:

a) This is long enough for most images.

b) For any image requiring longer Huffman codes the compression ratio is not good.

5.1.5.1 Adaptive Huffman-Coding For Code Book Generation

An arbitrary set of images, the training set, is processed as above to generate a standard code book. The input symbols occurring with higher frequency in these images are assigned shorter codes. A number of longer codes are fixed in advance for new symbols which will occur in other images, the testing set. Here, we assume that most symbols in the testing images can be found in the standard code book and only a few symbols need to be treated as new.

The compression ratios resulting from this code book approach are dependent on the symbols selected from the training set. Any uncommon symbols in the code book will lengthen the codes for all the new codes. The number of symbols in a training set of ten images ranges from 1100 to 3100 depending on the system scale factor (factor_K) and the particular images. The number of new symbols may range from 2500 to 8000 depending on the referent code book. This system only handles those images which present fewer new symbols than the number of codes available in the code book.

5.2 Approach

Figure 5-1 is a block diagram of the system. In addition to the provision for *sending vectors not in* the codebook we have included the capability of sending error bits which give additional significant bits to correct the output of the vector quantization.

The wavelet transform step may use either separable or non separable filters. As in Antonini's report, we have implemented a version of the quincunx pyramid for the non separable case. These filters produce a decomposition with a multi-resolution scale factor of $\sqrt{2}$. Other non-separable filters are also possible, as reported by Lawton and Resnikov [19] and Gröchenig and Madych [14] but have not yet been investigated. One curious aspect of some of these filters is that the domain of support of the associated wavelet is a set with a fractal boundary, which can be used to give a periodic tiling of the plane.

In our present system we perform levels of transform in the quincunx pyramid. This replaces the original image by a set of "super blocks" of wavelet coefficients of different types. Because six levels are performed this gives 256 types of coefficients each type corresponding to a different series H to G operations (as identified in the background section). For example, one set of coefficients would correspond to HHHHHHHH and a second to HHHHHHHG a third to HHHHHHGH and so forth.

This has the effect of replacing an original 512 x 512 image by 4096 blocks of 64 coefficients each. I.E. there are 4096 coefficients of a given type. Statistical analysis of the various types of coefficients indicates that most of the original image intensity will be concentrated in a few types of coefficients, one of which is the coefficient arising from the HHHHHHHH operation. The statistical variation is correspondingly greater of the coefficients which concentrates the image intensity, therefore these coefficients contain most of the information on the original image. Table 5-2 gives a statistical analysis of the coefficients from an 8-level transform of the Lena Image. By keeping the information in these highly-variable coefficients and throwing away the information in the other coefficients, we can obtain respectable compression of the image, without too much degradation. A computation of the standard deviations of the wavelet coefficients forms the basics of our adaptive quantization. We set the number of bits to be preserved in the vector quantization by scaling each coefficient according to the standard deviation of its particular class. This process is illustrated in Figure 5-1.

```
========= SDj: sd(16 x 16) ===========
 0  40.8    .4    .8    .2   2.3    .3    .7    .2   5.2    .3    .8    .2   1.7    .2    .6    .2
 1    .2    .6    .3    .6    .2    .5    .2    .4    .2    .4    .2    .5    .2    .3    .2    .3
 2    .6    .3   1.2    .3    .9    .3   1.5    .3    .6    .3   1.1    .3    .7    .2   1.0    .2
 3    .4    .4    .3    .8    .3    .5    .2    .5    .3    .4    .3    .6    .2    .4    .2    .3
 4   2.2    .3   1.0    .2   3.3    .4    .9    .2   2.2    .3    .8    .2   3.5    .2    .7    .2
 5    .3    .6    .2    .4    .3    .6    .3    .5    .2    .4    .2    .3    .1    .3    .2    .4
 6   1.4    .3   1.5    .3   1.0    .3   1.9    .3   1.0    .2   1.2    .2    .6    .2   1.3    .2
 7    .3    .5    .2    .5    .4    .6    .3    .7    .3    .5    .2    .4    .3    .4    .2    .6

 8   5.4    .3   1.0    .3   2.4    .2    .6    .2   9.5    .4    .9    .2   2.2    .3    .7    .2
 9    .2    .4    .3    .6    .2    .3    .2    .3    .2    .5    .2    .5    .2    .4    .2    .4
10    .9    .3   1.5    .3    .7    .2   1.2    .2    .7    .3   1.3    .3    .8    .2   1.3    .2
11    .4    .6    .3    .7    .3    .4    .2    .4    .4    .4    .3    .7    .3    .4    .2    .4
12   3.1    .3    .9    .2   3.4    .2    .8    .2   2.5    .3   1.0    .2   4.5    .3    .9    .2
13    .2    .5    .2    .4    .2    .4    .2    .5    .3    .5    .2    .4    .2    .5    .3    .6
14   1.0    .3   1.5    .3    .7    .3   1.3    .3   1.1    .3   1.7    .3    .8    .3   1.9    .3
15    .3    .5    .2    .5    .3    .5    .3    .6    .3    .5    .2    .5    .4    .5    .3    .8
```

Table 5-2 Statistical Analysis of Wavelet Coefficients

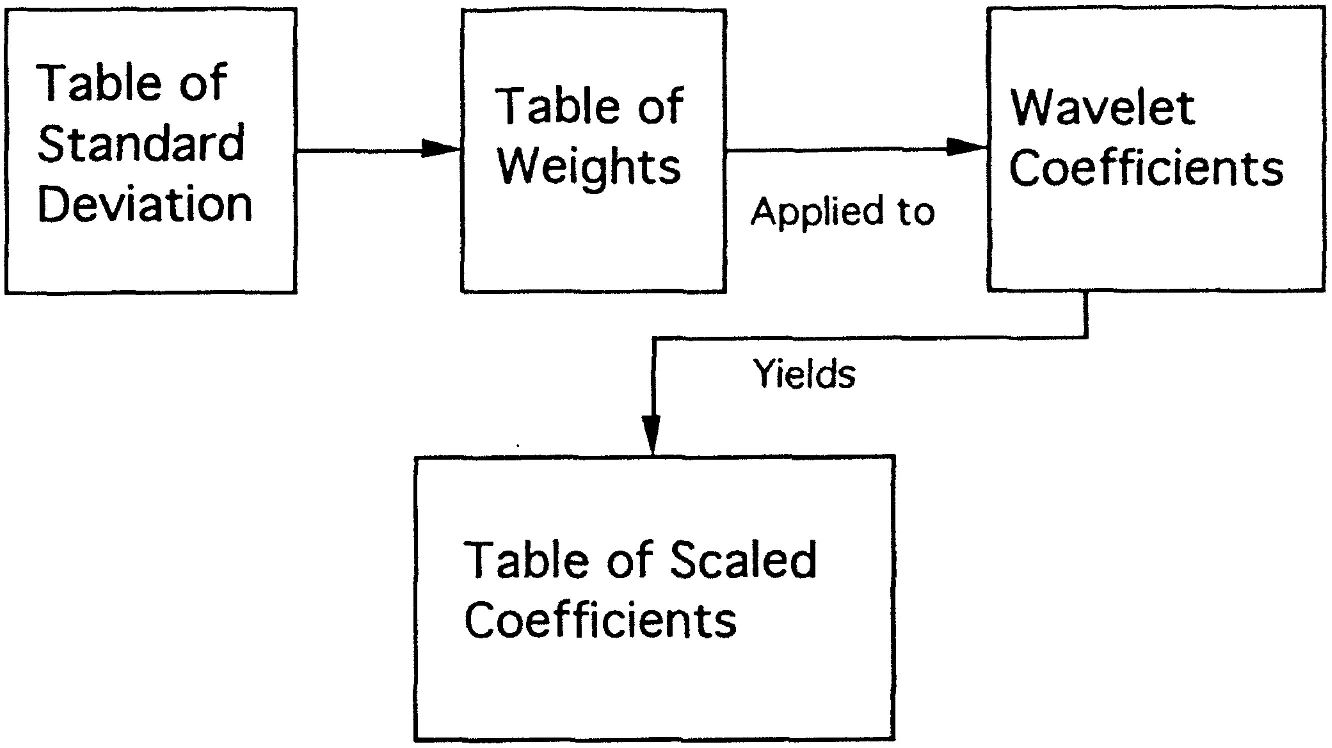


Figure 5-1 Adaptive Process for Scaling Wavelet Coefficients

In particular, by scaling a class of insignificant coefficients by a factor which can be expected to yield a number less than one half, then truncating to the nearest integer we effectively set all entries of that class to zero. For the vector quantization step which we employ in the present work the scaled coefficients are divided into 16 groups (or blocks of coefficients).

We have investigated D^{16} and Λ^{16} lattices in the present connection. General methods for constructing such lattices may be found in Conway and Sloan [7].

Note that the output of the blocks of entries subsequent to scaling and vector quantization can be regarded as vectors in a 16-dimensional space. Because the pattern of significant wavelet coefficients is similar in each block of 16, we expect zeros to consistently appear in some of the entries. This has the effect of reducing the dimension of the set of output vectors. Because of this and because the entries in the vectors are bounded we are left with a small class of vectors likely to appear. Because we employ a Huffman coding step prior to storing the compressed image this has the great benefit of keeping the codebook size small.

In addition to implementing data compression and decompression schemes as shown in Figure 5-2 we also developed software for using an arbitrary set of images to generate a codebook. This software consists of wavelet transform, statistical analysis, scaling vector quantization and adaptive Huffman coding modules. The adaptive H-coding algorithm adds new vectors to the codebook as they are encountered in each new image process. We take the codebook as stable when each new image adds less than some predetermined percentage (usually < 1%) of new entries to the codebook. A stable codebook may be used as a "standard codebook".

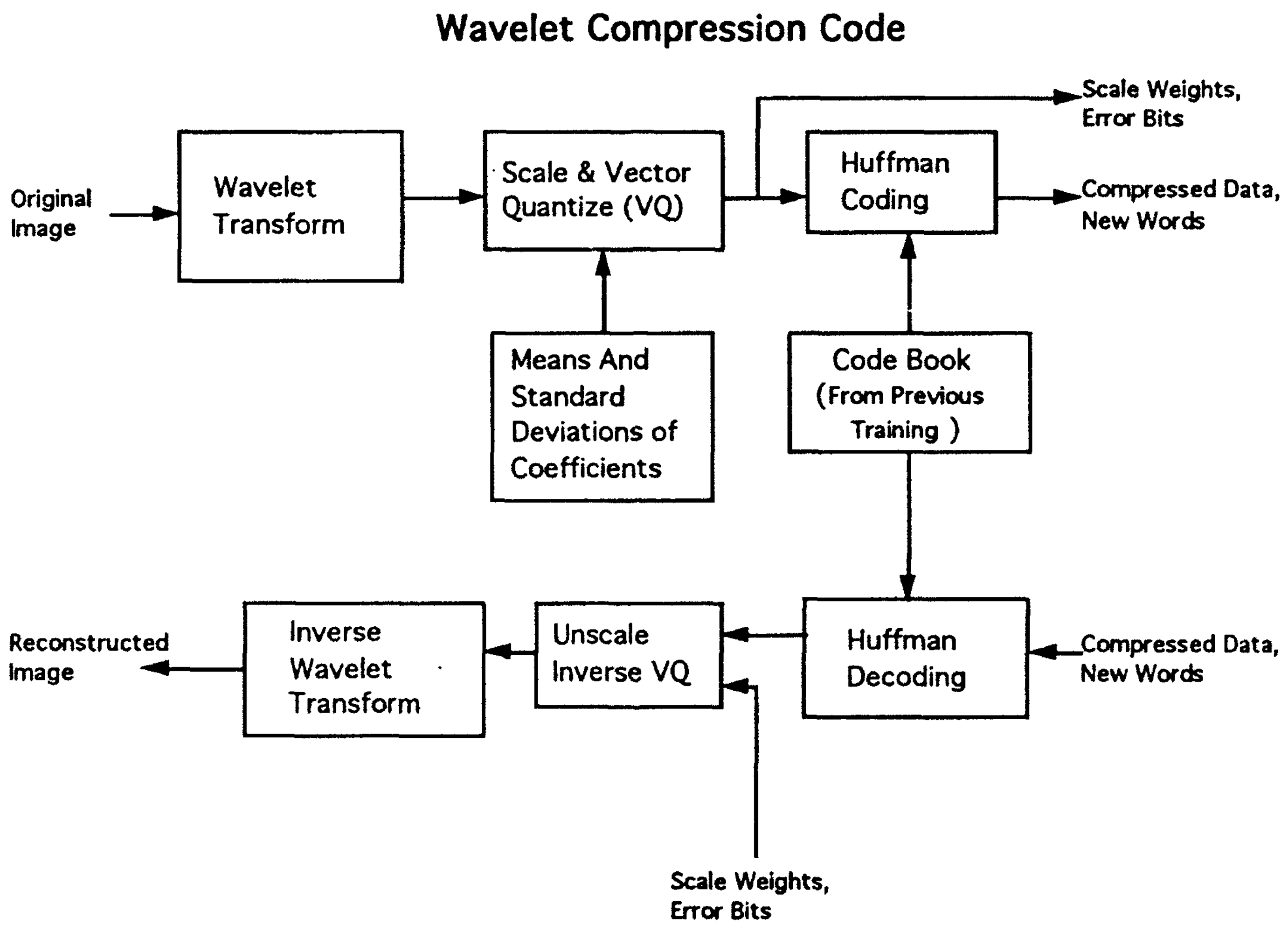

Wavelet Compression Code
Original Image
Wavelet Transform
Scale & Vector Quantize (VQ)
Huffman Coding
Scale Weights, Error Bits
Compressed Data, New Words
Means And Standard Deviations of Coefficients
Code Book (From Previous Training)
Reconstructed Image
Inverse Wavelet Transform
Unscale Inverse VQ
Huffman Decoding
Compressed Data, New Words
Scale Weights, Error Bits

Figure 5-2 Wavelet Compression Code

5.3 Accomplishments

The major accomplishment of our research into wavelet methods was the development of an adaptive scheme for wavelet compression and an adaptive scheme for developing a codebook (as described in the previous section).

Up to the present, our investigations have been limited to a selection of 128 x 128 and 512 x 512 greyscale images, but the results are promising. A typical compression run on both sizes of images produces a 12:1 compression ratio or better at a 30 dB peak signal-to-noise ratio. The visual quality of some types of reconstructed images is very good; in fact the error image must be scaled up to be visible on our computer screen under ambient light. Good image quality was preserved for both faces and fingerprints at the 12:1 compression ratio. Typical times for encoding a 128 x 128 image are about 5 minutes and decoding about 3 minutes on a 486 processor. We have not yet attempted to optimize the code.

Figures 8 and 9 in Appendix 2 show images of a fingerprint compressed and decompressed by means of our wavelet code. Note that the artifacts in the decompressed images are much less apparent than the artifacts in images reconstructed from the fractal code. These figures also show fingerprints coded and reconstructed by means of fractal techniques. Note that the artifacts in the fractal reconstruction appear to be "blocky", while the artifacts in the wavelet reconstruction appear to be induced by edge effects.

6.0 Conclusions

NETROLOGIC's work in image compression resulted into two complete software programs for the PC environment where both fractals and wavelets were successfully employed. Each approach has subsets of images where advantages accrue. In general wavelets can provide better quality at high compression ratios, but fractals enable fast and simple decompression.

6.1 Potential Dual-Use Follow On

NETROLOGIC has entered a joint venture with CST Images to implement the wavelet code developed in this program with an I860 accelerator board. In addition, we received a request to use our techniques to compress a movie segment and are setting up a CRDA with NRD in San Diego. We entered discussions with a major oil company to apply our compression methods to compression seismic logs which currently occupy 100,000 magnetic tapes.

References

[1] M. Antonini, M. Barlaud and P. Mathieu, "Image Coding Using Vector Quantization of Wavelet Coefficients", *Proc. IEEE Int Conf.* ASSP, Toronto Canada, 2273, May (1991).

[2] M. Barnsley and A. D. Sloan, "A Better Way to Compress Images", *Byte*, January (1988).

[3] M. Barnsley, *Fractals Everywhere*, Academic Press, San Diego, (1989).

[4] G. Beylkin, R. Coifman and V. Rokhlin, "Fast Wavelet Transforms and Numerical Algorithms", *Comm. Pure Appl. Math*, **44**:141-183 (1991)

[5] R.D. Boss and E.W. Jacobs, "Fractal-Based Image Compression", NOSC Technical Report 1315, September 1989. Naval Ocean Systems Center, San Diego, CA 92152-5000.

[6] R.D. Boss and E.W. Jacobs, "Fractal-Based Image Compression II", NOSC Technical Report 1362, June 1990. Naval Ocean Systems Center, San Diego, CA 92152-5000.

[7] J. H. Conway and N. J. A. Sloane, "Voroni Regions of Lattices, Second Moments of Polytopes and Quantization", *IEEE Transactions on Information Theory,* **28**:211, (1982)

[8] J. H. Conway and N. J. A. Sloane, "Fast Quantizing and Decoding Algorithms for Lattice Quantizers and Codes", *IEEE Transactions on Information Theory,* **IT-28**(No. 2):227, (1982).

[9] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, Springer-Verlag, New York (1988)

[10] S. Demko, L. Hodges and B. Naylor, "Construction of Fractal Objects with Iterated Function Systems", *Computer Graphics,* **19**:271, (1988).

[11] Y. Fisher, E.W. Jacobs and R.D. Boss, "Fractal Image Compression Using Iterated Transforms", to appear in *Data Compression*, J. Storer, Editor, Kluwer Academic Publishers, Norwall, MA.

[12] Y. Fisher, E.W. Jacobs and R.D. Boss, "Fractal Image Compression Using Iterated Transforms", NOSC Technical Report, Naval Ocean Systems Center, San Diego, CA 92152-5000.

[13] Y. Fisher and A. Lawrence, "Fractal Image Compression for Mass Storage Applications", *SPIE Proceedings (Image Storage and Retrieval Systems),* **1662**:244, (1992).

[14] K. Gröchenig and W. R. Madych, "Multiresolution Analysis, Haar Bases, and Self-Similar Tilings of R^n", *IEEE Transactions on Information Theory,* **38**:556, (1992).

[15] E.L. Hall, *Computer Image Processing and Recognition* , Academic Press, New York (1979).

[16] J. E. Hutchinson, "Fractals and Self Similarity", *Indiana University Mathematics Journal,* Vol.35, No.5, (1981).

[17] A. Jacquin, "A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding", Doctoral Thesis, Georgia Institute of Technology, (1989).

[18] E. W. Jacobs, Y. Fisher and R.D. Boss, "Image Compression: A Study of the Iterated Transform Method", to appear in *Signal Processing.*

[19] W. M. Lawton and H. L. Resnikoff, "Multidimensional Wavelet Bases", Preprint, Aware, Inc., Cambridge, Mass, (1991).

[20] S. Mallat, "Multiresolution Approximation and Wavelets", *Trans. Amer. Math. Soc.*, **315**:69-88 (1989)

[21] H. O. Pietgen and D. Saupe, Editors, *The Science of Fractals*, Springer Verlag, New York (1989).

[22] H. O. Pietgen, D. Saupe and H. Jurgens, Editors, *Fractals for Class Room,* Springer Verlag, New York (1991).

[23] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Second Edition, Volume 1, Academic Press, Florida (1982).

[24] W. Rudin, *Real and Complex Analysis*, McGraw Hill, New York. (1974).

[25] E. Walach and E. Karnin, "A Fractal Based Approach to Image Compression", *Proceedings of ICASSP*, Tokyo (1986)

[26] L. Domash, "Optical Computer for Fractal Image Analysis", Navy Report No. NAV-0238-FM-9179-444, March (1991).

[27] V. Ryan, "Preliminary Results for Image Compression via Fractal Analysis", Foster-Miller (Preprint), June (1992).

Appendix 1 : Description of Algorithms Developed at Netrologic

Fractal Encoding:

Summary: An image is partitioned recursively into rectangles. Matches between similar portions of the image are found, and these matches determine self-transformations of the image which encode it as a fixed point of an operator in a space of all possible images.

```
* Select a fidelity tolerance $E$ ($E$ typically ranges between 4 and 8).
* Create a list of rectangles $R$. Set $R_1$ to contain all the pixels in the initial image $W$.
* Repeat until there are no rectangles in $R$.
        * Select the largest rectangle in $R$. Call it $R_i$.
        * Find the best transform for $R_i$. Call it $W_i$.
                * Create a domain pool $D$. A domain pool consists of rectangles chosen from an image. Domain size is d from the range size. Each dimension is multiplied by 2 or 3 (e.g. an 4x5 range is compared to domain sizes: 8x10, 8x15, 12x10, 12x15). The orientation of the domains can be opposite that of the range (e.g. an $m$ x $n$ range is compared to $m'$ x $n'$ and $n'$ x $m'$ domains). Range pixels are accessed left-to-right top-to-bottom; domain pixels can be accessed left-to-right top-to-bottom, left-to-right bottom-to-top, right-to-left top-to-bottom, left-to-right bottom-to-top.
                * Search through $D$ for the domain which has the least $L^2$ (rms) difference between the domain and $R_i$. Call it $D_i$.
```

The distance between the transformed domain and the range is measured by some metric, such as the L^n metric.

$$L^n = \delta_n(R_i, W_i(D_i)) = \frac{\left\{ \sum_{j=1}^{l} \sum_{k=1}^{m} \left[(scale_i * D_{i_{jk}} + offset_i) - R_{i_{jk}} \right]^n \right\}^{1/n}}{l^{1/n} * m^{1/n}} \qquad \text{(App. 1-1)}$$

To minimize the error [the distance between R_i and $W_i(D_i)$], compute the following:

$$s_R = \sum_{j=1}^{l} \sum_{k=1}^{m} R_{i_{jk}} \qquad \text{(App. 1-2)}$$

$$s_D = \sum_{j=1}^{l} \sum_{k=1}^{m} D_{i_{jk}} \qquad \text{(App. 1-3)}$$

$$s_{R^2} = \sum_{j=1}^{l} \sum_{k=1}^{m} R^2_{i_{jk}} \qquad \text{(App. 1-4)}$$

$$s_{D^2} = \sum_{j=1}^{l} \sum_{k=1}^{m} D^2_{i_{jk}} \qquad \text{(App. 1-5)}$$

$$s_{RD} = \sum_{j=1}^{l} \sum_{k=1}^{m} R_{i_{jk}} * D_{i_{jk}} \qquad \text{(App. 1-6)}$$

$$area = m * l \qquad \text{(App. 1-7)}$$

$$det = (area * s_{D^2}) - (s_D * s_D) \qquad \text{(App. 1-8)}$$

if (det==0.0) {scale=0; offset=s_R/area;} (App. 1-9)

else {scale=((areas_{RD}) - s_R * s_D))/det; offset=(s_R-(scale * s_D))/area;}* (App. 1-10)

$$rms = \sqrt{\frac{s_{R^2} - \left(\frac{s_R * s_R}{area}\right) - \left(scale * \left(s_{RD} - \left(\frac{s_R * s_D}{area}\right)\right)\right)}{area}} \qquad \text{(App. 1-11)}$$

Values of the affine regression parameters *scale* and *offset* must be adjusted while calculating *rms* because compressed output restricts their precision. *Scale* must be adjusted before computing *offset* and *offset* must be adjusted before calculating *rms*.

* Accept transform or simplify and loop.

* If the rms difference between D_i and R_i is less than E, store R_i, D_i, *scale*, and *offset* in W_i.

* Else Partition R_i: Remove the rectangle R_i from the list R and add R_i's two partitions to the list R. The partition position is determined by a standard edge detection scheme weighted by a tent map. The tent map is f(x) = |1 - x|, so that [-1,1] fits in the rectangle along the direction of partitioning. The edge detection

- scheme consists of finding the biggest absolute difference between successive row or column sums.

* Output the list W with as much compression as possible.

Decoding:

Summary: Decoding operates by iterating in the space of all images. Each transform W_i determines how pixels in an image should map to other pixels in the image. Iterating these maps converges to the fixed point image F.

```
* Read the list of transforms called $W$.
* Select a number of decoding steps $N$. $N$ is image dependent. We have found that after 3 iterations there is no perceived improvement, thought the fixed point $F$ rms continues to improve.
* Create the fixed point image $F$. Its initial contents do not matter.
* Repeat N times:
    * Apply each transform $W_i$ to $F$. The domain pixels are read from $F$, the range pixels are written to $F$.
* Output the image $F$.
```

References:

Y. Fisher, E.W. Jacobs, R.D. Boss, *Iterated Transform Image Compression*, NOSC Technical Report 148, NOSC, San Diego, April (1991).

Y.Fisher, *Fractal Image Compression*, **Fractals and Chaos**, H.O. Peitgen, D. Saupe, H. Jurgens, Appendix A, Springer Verlag, New York, (1992).

Color Images

Our fractal compression system (rc, rcdec, ...) can compress color images by converting them to gray-scale images and converting the reconstituted gray-scale images into a reconstructed color image.

Colors displayed on our computers are defined by their Red, Green, and Blue (RGB) components. Our systems display 64 different levels (shades, gray-scales) of Red, Green, and Blue independently. This means we have 6 bits for each color gun, for a total of 18 bits.

Colors are saved in files in many formats. We use mostly two types: a 24-bit RGB, and an 8-bit RGB color map. A 24-bit RGB file contains 24 bits of color information per pixel. Each color is specified by 8 bits. Since we can only display 6 bits of color resolution, these images are a slight loss of information however imperceptible . An 8-bit RGB color map, contains both a list of colors, specified by 24 bits, and an index into this list for each pixel in the image. The index is limited to 8 bits, so there can be a maximum of 256 different colors in the image.

Color images of the following formats, among others, can be used:

- Suffix -	- Description -
.RAS or .SUN	8-bit RGB color map or 24-bit RGB .SUN
.R, .G, .B	8-bit gray-scale Red, Green, and Blue components

Color images are converted to three gray-scale images representing the Y (Luminance), I (In-phase), and Q (Quadrature) components. For improved compression ratios the I and Q images are created at 1/2 of the original dimensions (1/4 of the area). Each of these images is compressed independently.

When regenerating the color images, each YIQ component image is decompressed, then they are combined into a single color image by an inverse of the process which dissected the color original.

Slight errors in any one of the YIQ components can lead to drastic color changes, so it is necessary to fine-tune the decompression color output. A 24-bit RGB color map file (.map) can be used to restrict which RGB values can be output by the color-reconstruction algorithm.

Appendix 2 : Images

Table of Contents

Note:
All images in this report are 8-bit gray-scales. Each pixel represents one of 256 gray levels. RMS calculations are also based on 256 possible levels.

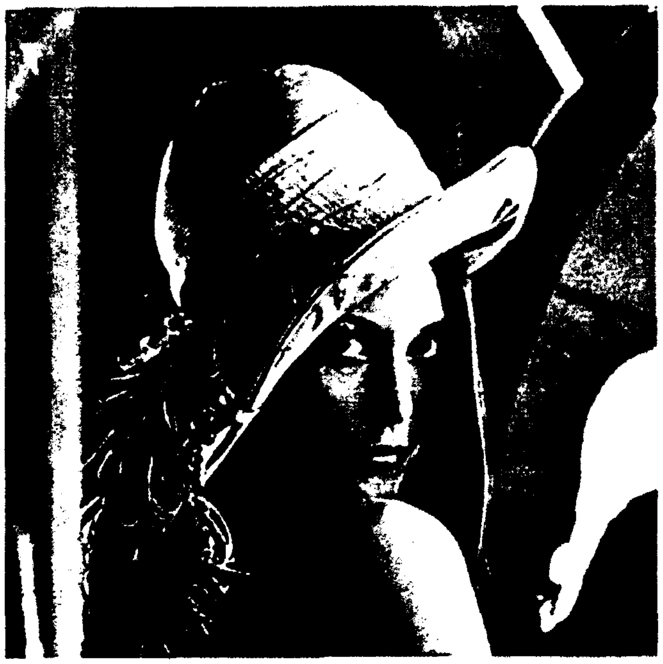


Image 1A Lena 512 x 512 Original

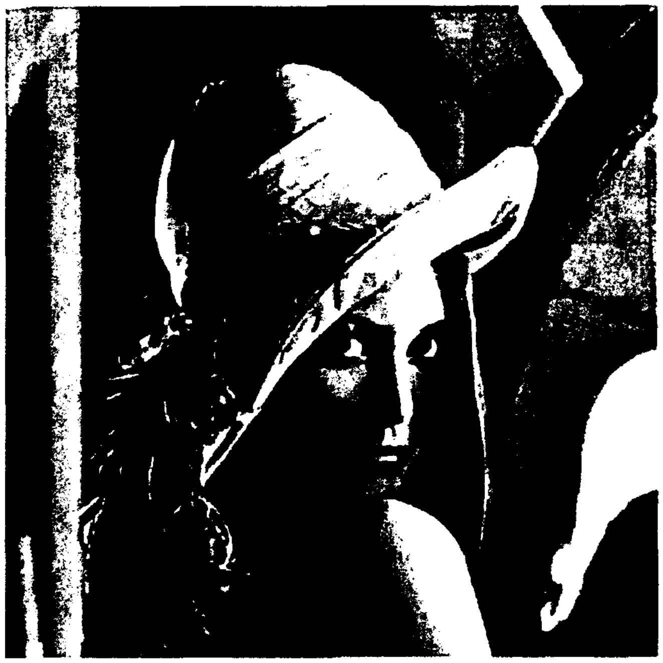


Image 1B Lena 512 x 512 Decompressed - HV Fractal Method

Compression Ratio = 27.014 S/N Ratio = 32.3

Compression Time = 44.83 min. Decompression Time = 12.63 sec.

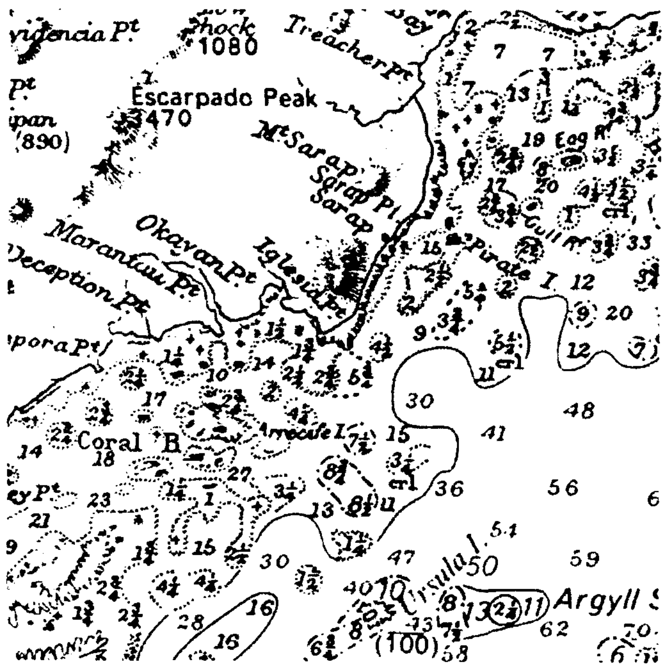


Image 2A Map 512 x 512 Original

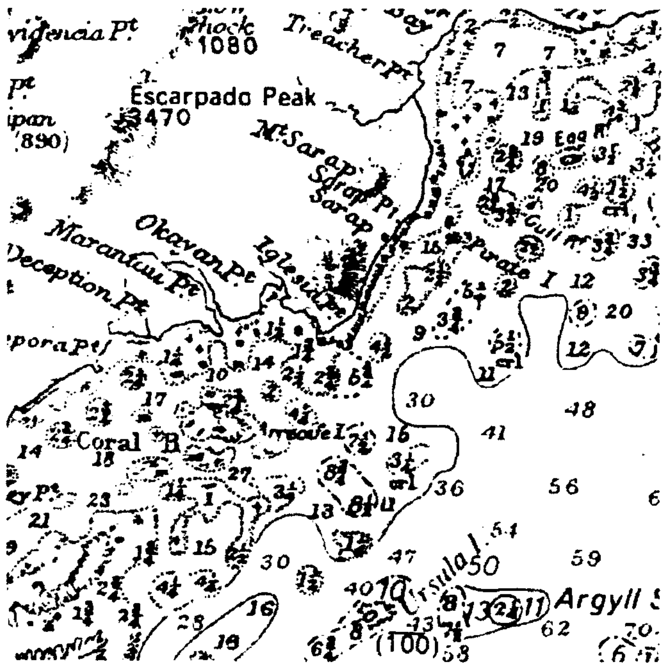


Image 2B Map 512 x 512 Decompressed - HV Fractal Method

Compression Ratio = 8.866 S/N Ratio = 24.74

Compression Time = 198.27 min. Decompression Time = 14.99 sec.

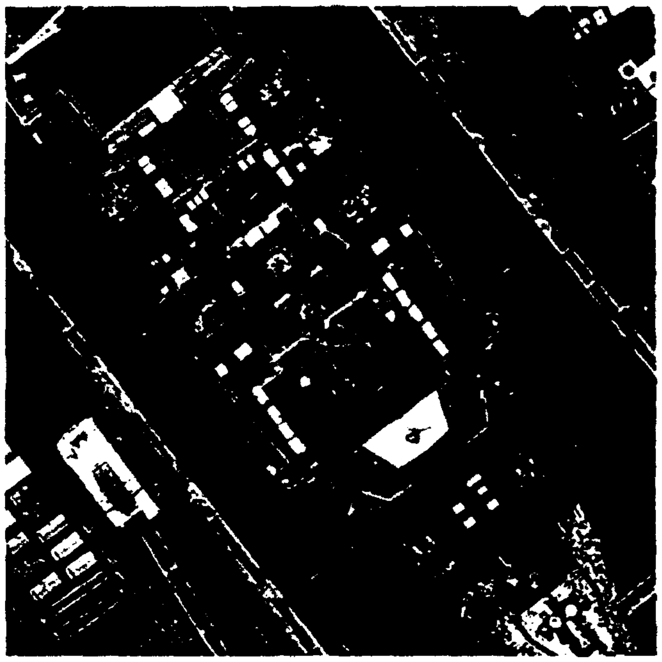


Image 3A Ship 512 x 512 Original

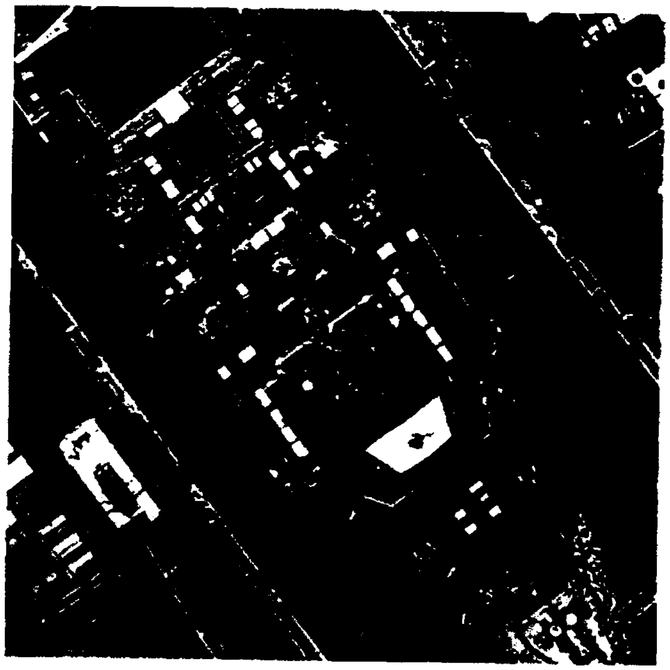


Image 3B Ship 512 x 512 Decompressed - HV Fractal Method

Compression Ratio = 5.066 S/N Ratio = 31.23

Compression Time = 238.8 min. Decompression Time = 17.52 sec.

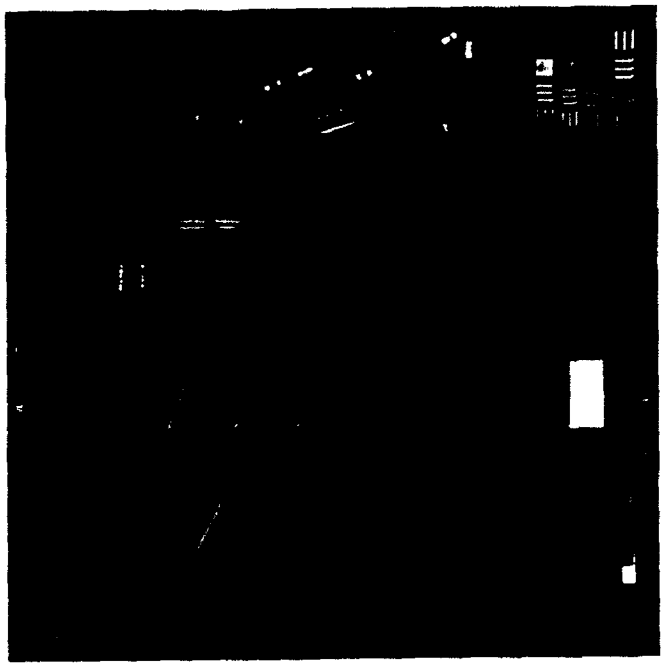


Image 4A Airfield 512 x 512 Original

Image 4B Airfield 512 x 512 Decompressed - HV Fractal Method

Compression Ratio = 23.279 S/N Ratio = 38.21

Compression Time = 31.6 min. Decompression Time = 13.02 sec.

Image 5A Grandfather & Granddaughter 512 x 512 Original

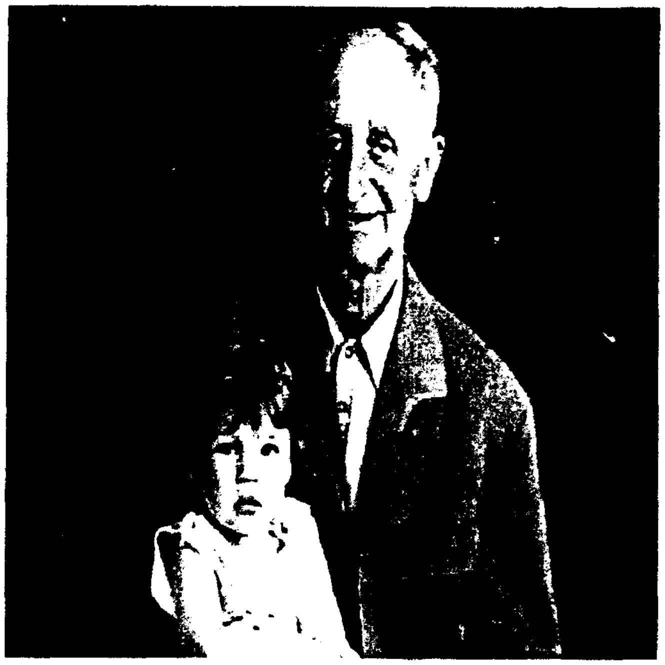


Image 5B Grandfather & Granddaughter 512 x 512 Decompressed - HV Fractal Method

Compression Ratio = 32.256	S/N Ratio = 39.35
Compression Time = 24.00 min.	Decompression Time = 12.46 sec.

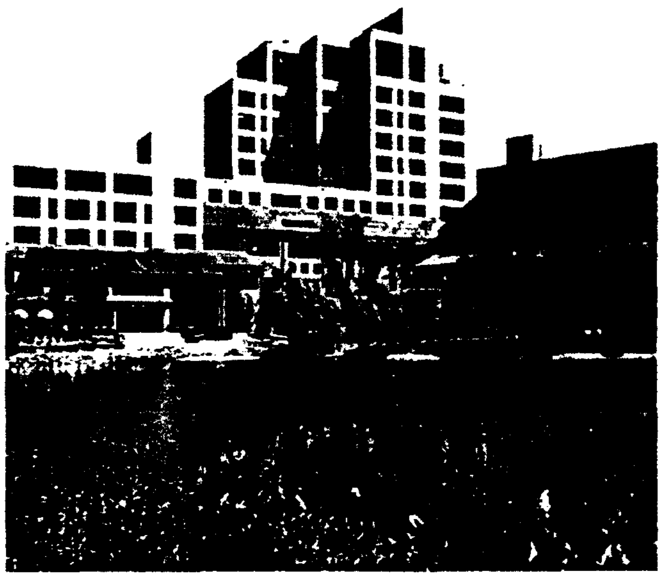


Image 6A Building 512 x 512 Original

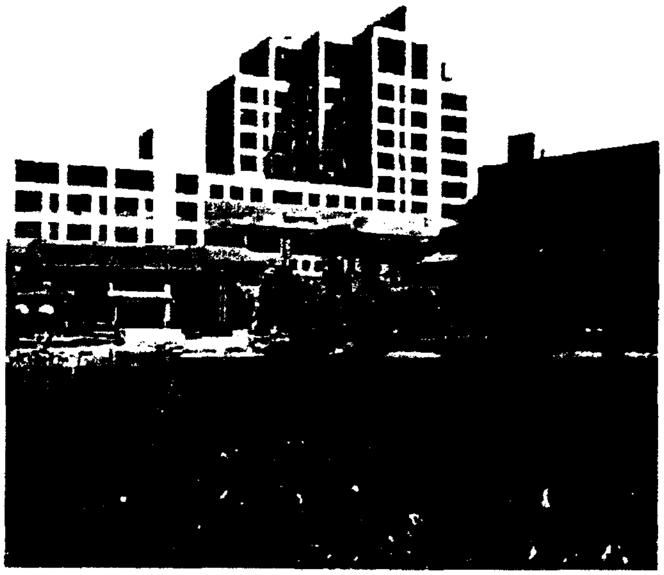


Image 6B Building 512 x 512 Decompressed - HV Fractal Method

Compression Ratio =	27.849	S/N Ratio =	29.11
Compression Time =	42.53 min.	Decompression Time =	12.75 sec.

Image 7 Fingerprint 512 x 512 Original

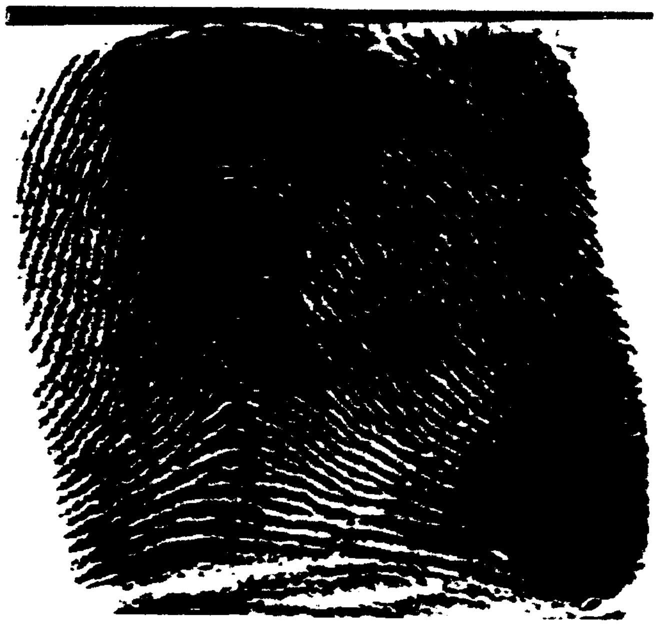


Image 8A Fingerprint 512 x 512 Decompressed - HV Fractal Method
Compression Ratio = 7.472 S/N Ratio = 33.33

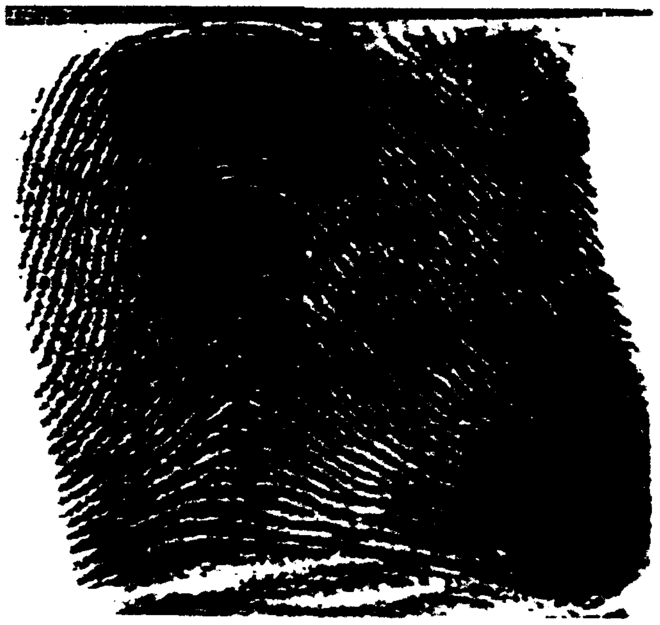


Image 8B Fingerprint 512 x 512 Decompressed - Wavelet Method
Compression Ratio = 9.17 S/N Ratio = 24.92

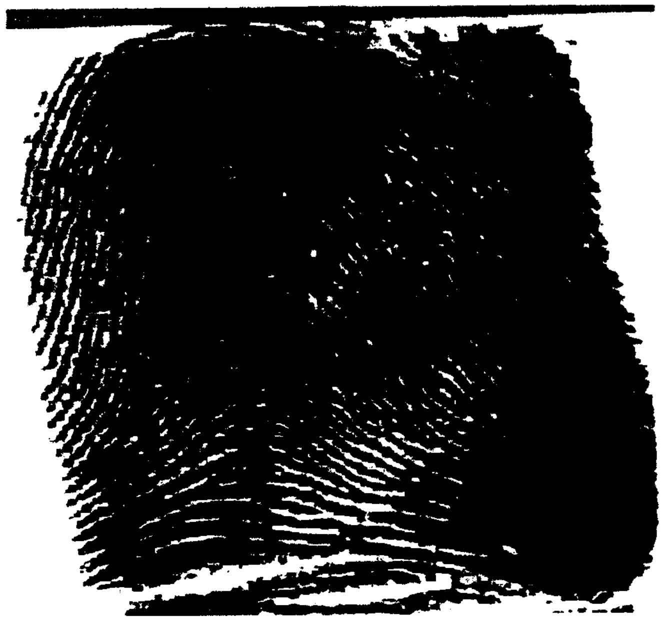


Image 9A Fingerprint 512 x 512 Decompressed - HV Fractal Method
Compression Ratio = 20.541 S/N Ratio = 27.38

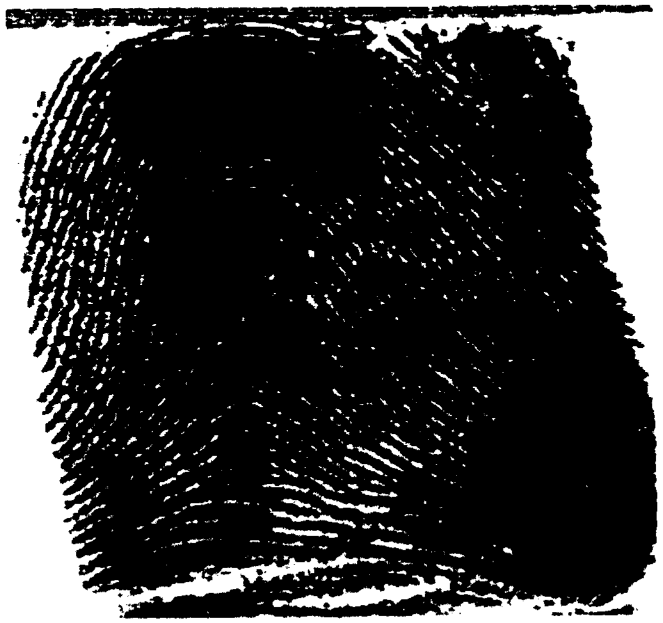


Image 9B Fingerprint 512 x 512 Decompressed - Wavelet Method
Compression Ratio = 23.32 S/N Ratio = 21.32